

30

LE JOURNAL QUI SE SAIGNE POUR SES LECTEURS

FEVRIER 1989

Ready
RUN APENDICE
Type mismatch
Ready
RUM "APENDICE"
Syntax error
Ready
KUN "APENDICE"
APENDICE.   not found
Ready

SERRE



EDITORIAL

Alors je prends quinze jours de vacances, et vous ne trouvez même pas le temps de programmer un clône de MULTIPLAN ou un super dBASE IV au lieu de regarder la roue de la fortune (sur Té-HèF-Hun)? Et moi, persuadé que je suis des performances de TURBO-Forth, convaincu d'avoir oeuvré pour les futures générations de programmeurs, je ne vois rien venir de vos grandes oeuvres. Allez-vous vous laisser déborder par le Futur Numérique Proche (institut informel dont je m'auto-proclame co-gérant avec moi-même, destiné à prendre le contre-pied du PAF)? Vous, les maîtres du TOKEN RING, dans quels méandres Basiciennes vous dispersez-vous?

SOMMAIRE

Nos coordonnées:
ASSOCIATION JEDI 17, rue de la Lancette 75012 PARIS
tel président: (1) 43.40.96.53
tel secrétaire: (1) 49.85.63.67 (SAM*JEDI, bal SECRETAIRE)
télétel: 3615 SAM*JEDI

PI
JUSQU'A 22000 DECIMALES




par J.L. SIRET

Systèmes: TURBO-Forth 83-Standard. Adaptable F83, optimisation assembleur pour PC et compatibles

Disponibilité: téléchargement par 3615 SAM*JEDI, prochainement en disquette dans le module 7 accompagnant TURBO-Forth.

— F

Ce programme calcule les 22000 décimales du nombre PI. Lancer le programme en tapant CALCULE; AFFPI affiche les 1000 premières décimales.

— GB

This program calculates the 22000 decimal ones of number pi. To launch the program while typing CALCULATES; AFFPI posts the first 1000 the decimal ones.

— D

Dieses Programm berechnet 22000 dezimal der Zahl PI. Das Programm in CALCULE; AFFPI tippend einzuführen schlägt 1000 erste dezimal an.

— NL

Dit programma berekent 22000 dezimal van het aantal PI. Het programma door CALCULE te typen te starten; AFFPI plakt 1000 eerste dezimal aan.

— SP

Este programa calcula el decimal 22000 unos del nu'mero pi. Para lanzar el programa mientras que el mecanografiar CALCULA; AFFPI fija los primeros 1000 el decimal unos.

LISTING: PI.FTH

```
EMPTY ONLY FORTH DEFINITIONS DECIMAL
\ Par PI/4 = 16 ATN(1/5) - 4 ATN(1/239)
\ Et ATN 1/5 = 1/5 - 1/3.5.5 + 1/5x5.5.5.5
\ - 1/7.5.5.5.5.5.5 ...
\ Quelques chiffres 1 min pour 1000 décimales si LG=505
\                   1 h 30 min pour 10000 décimales si LG=5005
\ Cette méthode ne permet que d'aller jusqu'à 22000
\ décimales car je ne travaille que sur un ^@64 K:
\ les segments suivants du forth.
\ Au fait, il faut signaler une erreur dans le fichier
\ FLOAT.FTH où les calculs étaient passés au pascal:
\ Eh bien le nombre PI a été amputé d'un chiffre:
\ c'est 3,14159265358979 ... 3238462433; il manquait un
\ 5 entre le 3 et le 8

505 CONSTANT LG HEX
0 DUP CONSTANT INV5 LG + DUP CONSTANT INV239
 LG + DUP CONSTANT COM2
 LG + DUP CONSTANT ATN5 LG + DUP CONSTANT ATN239
 LG + CONSTANT RESULT DECIMAL
: VARIABLES 0 DO VARIABLE LOOP ;    11
VARIABLES ESSD ADR1 ADR2 ADR3 NB SGN OCT COEF RET CPT SEG2
: VIDE SEG2 @ SWAP LG 0 LFILL ;
: /QR 0 OCT @ MU/MOD DROP ;
: R*/ *D RET @ 0 D+ OCT @ MU/MOD DROP ;
\ Adr1 + Adr2 --> Adr3
\ : ADD 0 RET ! 0 LG 1- DO ADR1 @ I + SEG2 @ SWAP LC@
\ ADR2 @ I + SEG2 @ SWAP LC@ RET @ + +
\       /QR RET ! ADR3 @ I + SEG2 @ SWAP LC! -1 +LOOP ;
ALSO
\ LABEL MODUL 0 # DH MOV 100 # AL CMP U>=
\ IF 100 # AL SUB 1 # DH MOV THEN RET C;
LABEL (ADD)
  SI PUSH DI PUSH DS PUSH LG # AX MOV AX CX MOV CX INC
         ADR1 #) BX MOV AX BX ADD BX SI MOV
         ADR3 #) BX MOV AX BX ADD BX DI MOV
         ADR2 #) BX MOV AX BX ADD
         SEG2 #) AX MOV  AX DS MOV AX ES MOV 0 # DH MOV
          STD
    HERE BYTE AL LODS 0 [BX] DL MOV DL AL ADD DH AL ADD
         0 # DH MOV 100 # AL CMP U>=
         IF 100 # AL SUB 1 # DH MOV THEN
         BYTE AL STOS BX DEC LOOP  CLD
         AX POP AX DS MOV AX ES MOV DI POP SI POP RET C;
CODE ADD (ADD) #) CALL NEXT C;
\ Adr1 - Adr2 --> Adr3
\ : SUB 0 RET ! 0 LG 1- DO ADR1 @ I + SEG2 @ SWAP
\ LC@ ADR2 @ I + SEG2 @ SWAP LC@ RET @ +
\      - DUP 0< IF OCT @ + 1 RET ! ELSE 0 RET ! THEN
\      ADR3 @ I + SEG2 @ SWAP LC! -1 +LOOP ;
LABEL (SUB)
   SI PUSH DI PUSH DS PUSH LG # AX MOV AX CX MOV CX INC
         ADR1 #) BX MOV AX BX ADD BX SI MOV
         ADR3 #) BX MOV AX BX ADD BX DI MOV
         ADR2 #) BX MOV AX BX ADD
         SEG2 #) AX MOV  AX DS MOV AX ES MOV 0 # DH MOV
         STD
   HERE BYTE AL LODS 0 [BX] DL MOV DH DL ADD DL AL SUB
         0 # DH MOV 0 # AL CMP <
         IF 100 # AL ADD 1 # DH MOV THEN
         BYTE AL STOS BX DEC LOOP  CLD
         AX POP AX DS MOV AX ES MOV DI POP SI POP RET C;
CODE SUB (SUB) #) CALL NEXT C;
\ Adr1 x Coef --> Adr2
: MULT 0 RET ! 0 LG 1-
   DO ADR1 @ I + SEG2 @ SWAP LC@ COEF @ R*/ RET !
      ADR2 @ I + SEG2 @ SWAP LC! -1 +LOOP ;
\ Adr1 : Coef --> Adr2
\ : DIV 0 RET ! LG  0 ?DO RET @ OCT @ *D ADR1 @ I +
\     SEG2 @ SWAP LC@
\        0 D+ COEF @ MU/MOD DROP ADR2 @ I + SEG2 @
\        SWAP LC! RET ! LOOP  ;
LABEL (DIV) SI PUSH DI PUSH DS PUSH LG # CX MOV
         ADR1 #) SI MOV   ADR2 #) DI MOV
         COEF #) BX MOV SEG2 #) AX MOV  AX DS MOV AX ES MOV
         CLD     CX DEC
   HERE BYTE AL LODS 0 # AL CMP 0= IF ROT BYTE AL STOS
 LOOP
         SI INC THEN SI DEC CX INC 0 # AX MOV
   HERE BX PUSH 100 # BX MOV BX MUL AX BX MOV BYTE AL LODS
   0 # AH MOV BX AX ADD
         0 # DX ADC BX POP BX DIV  BYTE AL STOS
         DX AX MOV  LOOP
         AX POP AX DS MOV AX ES MOV DI POP SI POP RET C;
CODE DIV (DIV) #) CALL NEXT C;
\ : DIV5   INV5   DUP ADR1 ! ADR2 !   5 COEF ! DIV ;
LABEL (DIV5) INV5 # AX MOV ADR1 # BX MOV AX 0 [BX] MOV
                          ADR2 # BX MOV AX 0 [BX] MOV
                5 # AX MOV COEF # BX MOV AX 0 [BX] MOV
                (DIV) #) CALL RET C;
CODE DIV5 (DIV5) #) CALL NEXT C;
\ : DIV239 INV239 DUP ADR1 ! ADR2 ! 239 COEF ! DIV ;
LABEL (DIV239) INV239 # AX MOV ADR1 # BX MOV AX 0 [BX] MOV
                              ADR2 # BX MOV AX 0 [BX] MOV
                239 # AX MOV COEF # BX MOV AX 0 [BX] MOV
                (DIV) #) CALL RET C;
CODE DIV239 (DIV239) #) CALL NEXT C;
: INIT HEIGHT 2* ALLOC DROP DSEGMENT HEIGHT + SEG2 !
        100 OCT ! INV5   VIDE 1 INV5 SEG2 @ SWAP LC!  DIV5
        SEG2 @ INV5 SEG2 @ ATN5   LG LCMOVE
        INV239 VIDE 1 INV239 SEG2 @ SWAP LC! DIV239
        SEG2 @ INV239 SEG2 @ ATN239 LG LCMOVE
        1 SGN ! 1 NB ! ;
\ : CH5 DIV5 DIV5 COM2 ADR2 ! ( COM2 VIDE ) NB @ COEF ! DIV
\       COM2 ADR2 ! ATN5 DUP ADR1 ! ADR3 ! SGN @ 1 =
\       IF ADD ELSE SUB THEN ;
LABEL CH5 (DIV5) #) CALL (DIV5) #) CALL
        COM2 # AX MOV ADR2 # BX MOV AX 0 [BX] MOV
        NB #) AX MOV  COEF # BX MOV AX 0 [BX] MOV
        (DIV) #) CALL
        COM2 # AX MOV ADR2 # BX MOV AX 0 [BX] MOV
        ATN5 # AX MOV ADR1 # BX MOV AX 0 [BX] MOV
        ADR3 # BX MOV AX 0 [BX] MOV
        SGN #) AL MOV 1 # AL CMP 0=
        IF (ADD) #) CALL ELSE (SUB) #) CALL THEN
        RET C;
\ : CH239
\  DIV239  DIV239 COM2 ADR2 ! ( COM2 VIDE ) NB @ COEF ! DIV
\        COM2 ADR2 ! ATN239 DUP ADR1 ! ADR3 ! SGN @ 1 =
\         IF ADD ELSE SUB THEN ;
LABEL CH239 (DIV239) #) CALL (DIV239) #) CALL
        COM2 # AX MOV ADR2 # BX MOV AX 0 [BX] MOV
        NB #) AX MOV  COEF # BX MOV AX 0 [BX] MOV
        (DIV) #) CALL
        COM2 # AX MOV ADR2 # BX MOV AX 0 [BX] MOV
        ATN239 # AX MOV ADR1 # BX MOV AX 0 [BX] MOV
        ADR3 # BX MOV AX 0 [BX] MOV
        SGN #) AL MOV 1 # AL CMP 0=
        IF (ADD) #) CALL ELSE (SUB) #) CALL THEN
        RET C;
\ : SUITE 2 NB +! -1 SGN @ * SGN ! CH5 CH239 ;
CODE SUITE CLI NB # BX MOV 2 # 0 [BX] ADD
```

```
        SGN # BX MOV 0 [BX] AX MOV 1 # DX MOV
        1 # AX CMP 0= IF -1 # DX MOV
        THEN DX 0 [BX] MOV CH5 #) CALL
        CH239 #) CALL STI NEXT C;
: CALPI ATN5  ADR1 ! RESULT ADR2 ! 16 COEF ! MULT
        ATN239 ADR1 ! COM2   ADR2 !  4 COEF ! MULT
       RESULT DUP ADR1 ! ADR3 ! COM2 ADR2 ! SUB ;
: AFFOCT 48 + EMIT 1 CPT +! CPT @ 5 MOD 0= IF 2 SPACES
       THEN CPT @ 50 = IF CR CPT OFF THEN ;
: AFFICHE
   DUP SEG2 @ SWAP LC@ 10 MOD 48 + EMIT ." ," CR CPT OFF
        LG 1  DO DUP  I +  SEG2 @  SWAP LC@ 10 /MOD AFFOCT
AFFOCT LOOP DROP CR ;
: AFFPI CALPI RESULT AFFICHE ;
: AFF5 ATN5 AFFICHE ;
: AFF239 ATN239 AFFICHE ;
: CHO INV5 0 BEGIN 2DUP + SEG2 @ SWAP LC@ 0= WHILE 1+
   REPEAT LG MIN NIP 2* ;
: CALCUL DARK INIT BEGIN SUITE    0 0 AT NB @ .
        KEY? IF KEY CASE 27 OF TRUE ENDOF
                         65 OF AFFPI FALSE ENDOF
                         68 OF CHO . CR FALSE ENDOF
                FALSE ENDCASE ELSE FALSE THEN UNTIL ;
\ Esc arrête les calculs
\ A affiche les décimales
    ( toutes et ça prend 2 min pour 10000 )
\ D affiche le nombre de décimales déjà calculées
```

■

FORTH

EDITEUR DE COMMANDES DOS

par J.L. SIRET

Système: TURBO-Forth.

Diffusion: téléchargement 3615 SAM*JEDI et prochainement module M7.

LISTING: DOSEDIT.FTH

```
\       ********************************
\       *  Editeur de commandes forth   *
\       *  et de chaines de caractères  *
\       *  compatible avec les touches  *
\       *  de fonction.                 *  par Jean-Luc SIRET
\       ********************************
\
\ OBJECTIF: C'est de pouvoir modifier une ligne de saisie
\ que ce soit une commande forth ou une chaine de caractères
\ avec les touches fléchées, les touches d'insertion et de
\ suppression. De plus, pouvoir rappeler d'anciennes
\ commandes forth à la DOSEDIT.
\ PRINCIPE: Tout ce qui passe par EXPECT doit être remanié.
\ CHAR     Dans le standard, lors d'une frappe de touches,
\          le curseur était toujours en dernière position et
\          la frappe d'un caractère normal ajoutait un
\          caractère en dernière position. Maintenant le
\          caractère tapé se met à la place de l'ancien si le
\          curseur se trouve au milieu de la zone de commande
\          et ne rajoute un caractère que s'il se trouve à
\          la fin.
\ CR-IN    Toute fin de saisie se termine par CR. Pour
\          conserver une trace de commandes, il faut donc
\          après garder en mémoire ces commandes dans une
\          zone que j'appelle DOSEDIT
\ DEL-IN   doit pouvoir supprimer n'importe quel caractère au
\          milieu de la zone de saisie
\ BACK-UP doit pouvoir effacer toute la ligne de la
\          position du curseur.
\
\ ETAPE 1 Tout d'abord il faut DEFERré dans le fichier
\          KERNEL.TXT les mots:
\  CR-IN , DEL-IN et BACK-UP les remplacer par (CR-IN)
\          (DEL-IN) et (BACK-UP)
\  Ne pas oublier de mettre plus loin dans la partie
\          renseignements des mots vectorisés:
\          ' (CR-IN) IS CR-IN de même pour les deux autres.
\  Rajouter VARIABLE CURS qui va conserver la position
\ courante du curseur avant le mot EXPECT et dans celui-ci
\ mettre son initialisation par CURS OFF
\     ... DUP SPAN ! SWAP 0  CURS OFF .....
\
\ ETAPE 2 Modification du second fichier NOYAU.TXT
\ Rajouter la possibilité d'exécuter une action par les
\ de contrôle actuelles mais leur code n'est pas un
\ caractère inférieur à 32 et c'est d'ailleurs une sequence
\ de deux codes 0 puis la touche .... comme les touches de
\ fonction. Ce ne sont pas des touches de fonction car leur
\ action doit être immédiate et non en mettant leur nom
\ dans la zone de saisie. D'où création d'une nouvelle
\ table d'action avant le mot FUNCTION
\ CREATE TABFL ] NOOP NOOP NOOP NOOP NOOP NOOP NOOP
\                NOOP NOOP NOOP NOOP NOOP NOOP [
\ Puis dans le mot FUNCTION juste après le mot KEY
\ écrire la ligne
\        DUP 71 83 BETWEEN IF 71 - 2* TABFL + PERFORM ELSE
\ et à la fin de ce mot  THEN ( 2 en tout) juste avant le ;
\
\ ETAPE 3 Recompiler le tout avec le METAcompilateur
\
\ ETAPE 4 Puis compiler la suite:
' (CR-IN) IS CR-IN ' (CHAR) IS CHAR \ par précaution lors
' (DEL-IN) IS DEL-IN ' (BACK-UP) IS BACK-UP \ d'une
\ deuxième compilation
EMPTY ONLY FORTH DEFINITIONS DECIMAL
100 CONSTANT CAPA
\ capacité du tampon de commandes 500 est plus confortable
\ mais j'ai toujours été radin en place
VARIABLE PTCR VARIABLE PTFL
\ pointeurs dans le tampon de commandes
VARIABLE CAREMP VARIABLE ADTIB BL CAREMP !
\ CAREMP est une variable qui la plupart du temps contient
\ un espace mais peut être remplacé par tout autre lorsque
\ l'on veut matérialiser la zone d'édition. Voir le mot ESS
\ à la fin par exemple
CREATE DOSEDIT CAPA ALLOT DOSEDIT CAPA ERASE
: ASSFL
  ' SWAP DUP 71 83 BETWEEN
  IF 71 - 2* TABFL + ! ELSE 2DROP THEN ;
\ Permet de brancher les actions des touches d'édition
: TIBL
   BEGIN DUP CAPA PTCR @ 1+ - < NOT WHILE
     0 BEGIN 1+ PTCR @ OVER - DOSEDIT + C@ OVER 1- = UNTIL
     DUP -1 * PTCR +! DOSEDIT PTCR @ + SWAP ERASE REPEAT ;
: ((CR-IN))    OVER TIB =
 IF BL SKIP DUP 0<>
   IF TIBL DOSEDIT 2DUP + 1+ PTCR @ CMOVE>
      2DUP DOSEDIT 1+ SWAP CMOVE DUP DOSEDIT C!
      DUP 1+ PTCR +!
   THEN
 THEN (CR-IN) PTFL OFF CURS OFF ;
' ((CR-IN)) IS CR-IN
\ Seules les commandes forth sont conservées d'où le
\ test OVER TIB =
\ Dans le tampon DOSEDIT les commandes sont rajoutées au
\ début et les anciennes décalées. Il y a calcul de la
\ place nécessaire et suppression des plus anciennes
\ commandes pour pouvoir entrer chaque nouvelle commande.
\ Une amélioration possible serait d'éviter les redondances
\ de commande et de ne garder que les différentes: (d'abord
\ comparaison etc...)
\ A la fin CR-IN reinitialise la variable CURS
: ((BACK-UP))
  CURS @ BACKSPACES DUP CAREMP @ REPLICATE
  BACKSPACES 0 CURS OFF ;
' ((BACK-UP)) IS BACK-UP
\ On remarquera la séquence  CAREMP @ REPLICATE  pour
\ conserver lors d'un effaçage les caractères de la zone
\ d'édition.
: CURS? CURS @ MIN CURS ! ;
\ Voilà un mot simple pour un problème qui m'a cassé les
\ pieds longtemps. Normalement chaque saisie se termine par
\ un CR, et OUT doit être réinitialisé avant la prochaine
\ saisie comme la variable CURS.
\ Et non, car la fin peut se faire aussi quand la zone est
\ saturée; quand on arrive au maximum que cette zone peut
\ contenir. A ce moment, les initialisations ne se
\ faisaient plus au même endroit et l'on pouvait arriver
\ à des aberrations comme le curseur en position 5 avec
\ seulement 2 caractères d'entrés. Chaque mot commence donc
\ par vérifier la concordance du curseur grâce à CURS?
: METIB
  COUNT DUP >R 2DUP TYPE ADTIB @ SWAP
  CMOVE R@ 1+ PTFL +! R> ;
: HTBAS DUP CURS? OVER ADTIB ! BACK-UP DROP ;
: FLHT HTBAS PTFL @ DOSEDIT + DUP C@ 0 <>
     IF METIB ELSE DROP 0 THEN DUP CURS ! ;
: CPTAR 0
  BEGIN 1+ PTFL @ OVER - DOSEDIT + C@ OVER 1- =
  UNTIL -1 * PTFL +! ;
: FLBAS HTBAS PTFL @ 0<>
```

```
    IF CPTAR PTFL @ DOSEDIT + METIB
    ELSE 0 THEN
  ELSE 0 THEN DUP CURS ! ;
\ Ceci pourrait suffire pour un DOSEDIT minimum
: ((CHAR))
  OVER CURS? 3DUP EMIT DROP CURS @ + C! CURS 1+!
  CURS @ MAX ;
' ((CHAR)) IS CHAR
\ CHAR écrit à la position du curseur et non forcément à la
\ fin. Si c'est la fin la chaîne vient donc d'augmenter
\ comme l'ancien CHAR
: FLGH DUP CURS? CURS @ 0<> IF BS EMIT CURS 1-! THEN ;
: FLDRT
  DUP CURS? CURS @ OVER <
  IF OVER CURS @ + C@ CURS 1+! EMIT THEN ;
: FLHOME DUP CURS? CURS @ BACKSPACES CURS OFF ;
: FLEND DUP CURS? CURS @ BACKSPACES 2DUP TYPE DUP CURS ! ;
\ Déplacement dans la zone de saisie
: FLINS
  DUP CURS? CURS @ OVER <
  IF DUP CURS @ - >R OVER CURS @ + DUP DUP 1+
     R> CMOVE> BL SWAP C! 1+ CURS @ BACKSPACES 2DUP TYPE
     DUP CURS @ - BACKSPACES
  THEN ;
: FLDEL. DUP CURS? CURS @ OVER <
  IF DUP CURS @ - >R OVER CURS @ + 1+ DUP 1-
     R> CMOVE 2DUP + 1- CAREMP @ SWAP C! CURS @
     BACKSPACES 2DUP TYPE
     DUP CURS @ - BACKSPACES 1-
  THEN ;
\ FLINS insère un espace à la position du curseur mais
\ FLDEL. supprime le caractère sous le curseur et diminue
\ la chaîne de 1. Le caractère de remplacement est mis à
\ la fin. Sur certains compatibles DEL. est obtenu
\ avec la touche ANNUL
: ((DEL-IN))
  DUP CURS? CURS @ 0<>
  IF BS EMIT CAREMP @ EMIT BS EMIT BL OVER CURS @ + 1- C!
     DUP CURS @ =
     IF 1- THEN
     CURS 1-!
  ELSE BELL EMIT THEN ;
' ((DEL-IN)) IS DEL-IN
\ Remplace l'ancien DEL-IN et supprime le caractère qui
\ précède le curseur et le remplace par un blanc. Si le
\ curseur se trouve en dernière position, la chaîne
\ est diminuée de 1
80 ASSFL FLBAS
\ On remplit le tableau TABFL par les actions
\ correspondantes
72 ASSFL FLHT
\ 80 est le code de la touche flèche haute etc
75 ASSFL FLGH
\ Il faut noter que ce tableau n'est pas entièrement rempli
77 ASSFL FLDRT  \ Les codes 74 76 et 78 ne servent à rien,
\ enfin chez moi
82 ASSFL FLINS
\ Les codes 73 et 81 de Pgup et Pgdn ne sont pas renseignés
83 ASSFL FLDEL. \ mais peuvent l'être.
71 ASSFL FLHOME
79 ASSFL FLEND
\ ---------------------------
\ Exemple pour la saisie d'une zone de caractère
20 STRING ESS$
: ESS
 DARK 15 5 AT ." Mot: ---------------------" ASCII -
 CAREMP !
 20 5 AT ESS$ INPUT$ BL CAREMP ! CR ESS$ TYPE ;
\ Pour ceux qui y voient un intérêt, tout ceci peut être
\ mis dans le fichier NOYAU.TXT et compilé avec le noyau et
\ protégé par l'habituel MARK EMPTY HERE FENCE !
```

UTILITAIRE DE MENUS DEROULANTS

par J.L. SIRET

Système: TURBO-Forth

Diffusion: téléchargement 3615 SAM*JEDI et prochainement module M7.

LISTING: UTILMENU.FTH

```
\ * Utilitaire: menus déroulants (encore un !) *
\ * par Jean-Luc SIRET                          *
\ *********************************************
\
\ Si l'ensemble de ce texte peut paraître décousu, cela est
\ normal puisque j'ai essayé de tirer quelques trucs d'un
\ programme écrit en forth par moi qui faisait au total aux
\ alentours de 50 Ko.
\ L 'interêt de ces menus déroulants est leur paramétrage
\ où la longueur, leur nombre, leur position et leur sens
\ ne se définissent qu'à l'utilisation. De plus, chaque
\ ligne du menu peut être documentée par un message
\ apparaissant sur la première ligne en haut de l'écran.
\ L'affichage se fait à l'écran grace au mot $>SCREEN
\ paru dans un JEDI et modifié pour qu'il puisse écrire
\ des chaînes extra-segment.
EMPTY ONLY FORTH ALSO DEFINITIONS DECIMAL
: VARIABLES 0 ?DO VARIABLE LOOP ;
VARIABLE VIDEO_RAM VARIABLE ATTR ALSO
CODE $>SCREEN
   DI POP   DI SHL   CX POP    AX POP    BX POP
     SI PUSH     AX SI MOV     DS PUSH
     VIDEO_RAM #) AX MOV      AX ES MOV
     ATTR #) AX MOV  BX DS MOV
     CLD    ( DO ) HERE     BYTE MOVS     AL STOS      LOOP
     DS POP      SI POP     DS AX MOV    AX ES MOV   NEXT C;
HEX
: INITVIO
  B800    800 40   10 LC@   30 AND   30 =   * +   VIDEO_RAM ! ;
DECIMAL
50 STRING A$
: AFFLG2
  0 SWAP AT EMIT 65 196 REPLICATE EMIT ;
 \ car2 car1 ligne
: AFFCL2
 OVER OVER 1+
 ?DO 2 PICK I AT 179 EMIT LOOP 2 PICK SWAP AT
 ROT EMIT AT EMIT ;   \ car2 car1 col lgn2 lgn1
80 STRING BLANC$
BLANC$ DROP 80 BLANK 80 BLANC$ DROP 1- C!
: EFFLG
  DSEGMENT SWAP BLANC$ ROT 80 * $>SCREEN ;
: EFFLS 1+ SWAP DO I EFFLG LOOP ;
: AFFLG= 0 SWAP AT 80 205 REPLICATE ;
8 VARIABLES LONGM COLM LGNM FLECH NBRM ADRM SENSM ADRMESS
: NORMAL 45 ATTR ! ; : INVERSE 112 ATTR ! ;
: NORMAL2 7 ATTR ! ;
: MODULO FLECH @ DUP NBRM @ >
  IF DROP 1
  ELSE DUP 1 <
    IF DROP NBRM @
    THEN
  THEN FLECH ! ;
: ADRECR 1- SENSM @ 0=
   IF 80 * ELSE LONGM @ * THEN
   LGNM @ 80 * + COLM @ + ;
: ADRN 1- LONGM @ * ADRM @ + ;
: AFF1' ADRN LONGM @ ; : AFF1 AFF1' TYPE ;
: AFF2' 1- 2* ADRMESS @ + @ COUNT ;
: AFF2 AFF2' TYPE ;
: AFF12 AFF1' -TRAILING TYPE SPACE AFF2 ;
: AFFMESS
  NORMAL2 0 0 AT 64 32 REPLICATE 1- 2* ADRMESS @ + @
  DSEGMENT SWAP COUNT 0 $>SCREEN ;
: AFFICHM
  DUP ADRN LONGM @ ROT ADRECR >R DSEGMENT -ROT R>
  $>SCREEN ;
: CHANGE
  NORMAL FLECH @ AFFICHM INVERSE FLECH +! MODULO
        FLECH @ DUP AFFICHM AFFMESS ;
: TOUCHE
   SENSM @ 0= DUP
   IF
     BEGIN DROP KEY2 DUP
       CASE    13 OF TRUE ENDOF
               27 OF TRUE ENDOF
              200 OF -1 CHANGE FALSE ENDOF
              208 OF  1 CHANGE FALSE ENDOF
               32 OF  1 CHANGE FALSE ENDOF
              FALSE SWAP
       ENDCASE
     UNTIL
     ELSE
       BEGIN DROP KEY2 DUP
         CASE 13 OF TRUE ENDOF
              27 OF TRUE ENDOF
             203 OF -1 CHANGE FALSE ENDOF
```

```
            32 OF   1 CHANGE FALSE ENDOF
           FALSE SWAP
        ENDCASE
      UNTIL
   THEN ;
: CADRE
  COLM @ 1- LGNM @ 1- 2DUP AT 187 EMIT
   LONGM @ 32 REPLICATE 201 EMIT  NBRM @ 0
   ?DO 1+ 2DUP AT 186 EMIT OVER
      LONGM @ + 1+ OVER AT 186 EMIT
   LOOP  1+ AT 200 EMIT LONGM @ 205 REPLICATE 188 EMIT ;
: MENCOM1  NBRM @ 1+ NORMAL 1
   DO I AFFICHM LOOP SENSM @ 0=
   IF CADRE THEN
   INVERSE FLECH @ DUP AFFICHM AFFMESS TOUCHE FLECH @ SWAP ;
: RENSPAR SENSM ! LGNM ! COLM ! FLECH ! ;
: MENCOM RENSPAR MENCOM1 ; \ sort avec choix et touche
: ENTMEN
   NBRM @ 0 ?DO HERE LONGM @ 1+ BLANK ASCII " WORD DUP
   1+ SWAP LONGM @ CMOVE  LONGM @ ALLOT LOOP ;
CREATE VIDE 1 C, 32 C,
: ENTMES HERE ADRMESS ! NBRM @ 0 ?DO VIDE , LOOP ;
: MESS
   ASCII " WORD SWAP DUP NBRM @ <=
   IF 1- 2* ADRMESS @ + ! HERE COUNT 1+ ALLOT
   ELSE DROP THEN DROP ;
: SORTMEN
   DUP C@ LONGM ! 1+ DUP C@ NBRM ! 1+ DUP @ ADRMESS !
   2+ ADRM ! ;
\ nbrm,longm , ---- longm , nbrm , adrmenu , pointe 1ere adr
\ des messages
: :DEFMEN
  CREATE DUP LONGM ! DUP C, OVER NBRM ! OVER C, * 2+
  HERE + , ENTMEN ENTMES DOES> SORTMEN ;

\ Explications des mots:
\ VARIABLES sert à déclarer plusieurs variables en meme
\ temps. Je le trouve plus lisible qu'une succession de
\ VARIABLE VAR1 VARIABLE VAR2 etc...
\ EFFLG (n ----  ) efface une ligne donnée grace à la
\ chaine BLANC$
\ EFFLS (n1,n2 --- ) effare un groupe de lignes de n1 à n2.
\ AFFLG= (n --- ) affiche une ligne de '=' à la ligne n.
\ KEY2 ( ---- car) sort le code de la touche frappée au
\ clavier en un seul code. Normalement par KEY la touche F10
\ sort deux codes: 0 puis 68
\ Dans KEY2 elle sort 68 + 128 = 196 directement.
\ Viennent ensuite le concept de menu définit par :DEFMEN
\  :DEFMEN  <nom>  (  Nombre,longueur  -------  (rien mais \
variables renseignées))
\ définit une nouvelle structure où sont compilées:
\    -la longueur de chaque ligne                      LONGM
\    -le nombre de barres de menu ( de lignes)         NBRM
\    -l'adresse de la première adresse des messages    ADRMESS
\    -les chaines de menu proprement dites             ADRM
\     pointe ici
\    -les adresses des messages proprement dits et non les
\     messages eux-mêmes pointant au départ sur VIDE (chaine
\     de 1 blanc)
\ Le mot MESS permet de rentrer les messages qui de longueur
\ variable (max 80) sont compilés à la suite du diction-
\ naire. Ces messages ne sont pas obligatoires.
\ ---------------------------------------------------------
\ Exemple  1 sans message
5 2 :DEFMEN VALID$ Oui"Non"
: OUI? 24 EFFLG 0 24 AT ." Confirmez-vous ?"  \
       1 65 24 VALID$ 1 MENCOM DROP 1 = 24 EFFLG ;
\ ---------------------------------------------------------
\ Exemple 2 tiré d'un exemple paru dans JEDI de M.ZUPAN
2 5 :DEFMEN IMP$ Ouvre"Ferme"
1 MESS Ouvre une redirection: l'imprimante sera passive"
2 MESS Ferme une redirection: l'imprimante redevient active"
\ IMP$ rend courant ce menu et renseigne les variables
\ associées.
\ Le mot MENCOM est le mot utilisateur qui permet d'utiliser
\ ce menu.
4 VARIABLES <FROM FROM> REDFL CHDEP
: FILENOM
   PAD 14 EXPECT PAD SPAN @ DUP 0=
   IF ASCII Z PAD C! 1+ THEN + 0 SWAP C! ;
: TO ?OPEN DUP <FROM ! DUP (FROM) ?DOS-ERR FROM> !
     ?OPEN PAD DUP 0 (SEARCH0)
     IF 1 (OPEN) ELSE 0 (CREATE) THEN ?DOS-ERR
     DUP 0 0 ROT 2 (SEEK) ?DOS-ERR 2DROP DUP TO> !
     SWAP (TO) ?DOS-ERR ;
: RESTORE
  FROM> @ DUP <FROM @ (IO) ?DOS-ERR (CLOSE) TO> @ (CLOSE) ;
: TO2       REDFL @ IF RESTORE THEN TO REDFL ON ;
```

```
: RESTORE2 REDFL @ IF RESTORE REDFL OFF THEN ;
: REDIR? 3 22 EFFLS 1 10 13 IMP$ 1 MENCOM 27 =
   IF DROP
   ELSE 1 =
     IF 0 15 AT ." Nom du fichier: " FILENOM 4 TO2
     ELSE RESTORE2 THEN
   THEN 3 22 EFFLS ;
\ avec les définitions correspondantes
\ La suite 1 10 13 IMP$ 1 MENCOM  s'explique ainsi.
\ le premier 1 indique le choix sur laquelle la barre en
\ inversion vient pointer au départ: Choix par défaut.
\   10 13 ( ligne colonne) donne le coin supérieur gauche
\ du menu.
\   le dernier 1 indique le sens du menu 0 = vertical et 1=
\ horizontal.
\ Ce nombre est d'ailleurs testé dans le mot MENCOM1 pour
\ encadrer un menu vertical.
\ Le mot MENCOM sort avec deux nombres: Le choix et la
\ touche de sortie pressée. Cette dernière est soit ESC ou
\ RETURN. A l'utilisateur de leur donner un sens.
\ Dans l'exemple, le cas ESC (27) permet d'ignorer le choix
(DROP) l'autre cas prend en compte ce choix: de 1 à 2 dans
\ l'exemple. Au fait les touches valides lors du choix sont
\ les flèches,ESC et ENTER. Flèches Gauche et Droite pour
\ horizontal et Haut Bas pour vertical.
\
\ ---------------------------------------------------------
\ exemple 3: Menus imbriqués horizontaux s'ouvrant sur des
\ verticaux:
\ ********************** Menu général **********
6             12                :DEFMEN               MENU1$
Choix1"Choix2"Choix3"Choix4"Choix5"Quitter"
6 MESS Appuyer sur ESC pour quitter
\ *********************** menu verticaux *******
3 13 :DEFMEN MENV1$ Ch11"Ch12"Ch13"
2 16 :DEFMEN MENV2$ Ch21"Ch22"
4 16 :DEFMEN MENV3$ Ch31"Ch32"Ch33"Ch34"
3 14 :DEFMEN MENV4$ Ch41"Ch42"Ch43"
2 14 :DEFMEN MENV5$ CH51"CH52"
\ Comme à l'issue de chaque choix horizontal doit se faire
\ un nouveau choix vertical. Il faut définir ces mots.
: MENV1 1 1 3 MENV1$ 0 MENCOM ;    \ Premier vertical
: MENV2 1 12 3 MENV2$ 0 MENCOM ;
: MENV3 1 24 3 MENV3$ 0 MENCOM ;
: MENV4 2 36 3 MENV4$ 0 MENCOM ;
: MENV5 1 48 3 MENV5$ 0 MENCOM ;
: MENV6 0 27 ;   \ quitter donc rien à faire !
6 CASE: MENVER MENV1 MENV2 MENV3 MENV4 MENV5 MENV6 ;
: ND 0 24 AT ." Action non disponible pour le moment."
  KEY DROP 24 EFFLG ;
\ Tous les choix verticaux sont alors répertoriés  dans une
\ matrice 10x6. Le nombre 10 est ici surdimensionné car il
\ correspond à 10 choix verticaux.
\    CHOIX:       0    1    2    3    4  5  6  7  8  9
\ bien sur il faudrait écrire les mots CH11 etc..
60 CASE: CHOIXACT CH11 CH12 CH13 ND   ND ND ND ND ND ND
                  CH21 CH22 ND   ND   ND ND ND ND ND ND
                  CH31 CH32 CH33 CH34 ND ND ND ND ND ND
                  CH41 CH42 CH43 ND   ND ND ND ND ND ND
                  CH51 CH52 ND   ND   ND ND ND ND ND ND
                  ND   ND   ND   ND   ND ND ND ND ND ND ;
\ Le choix de départ est 2 et chaque nouveau est gardé pour
\ que revenant d'un menu vertical, on aille bien au choix
\ horizontal correspondant.
: MENU
  INITVID 2 CHDEP ! BEGIN 2 AFFLG= CHDEP @ 0 1 MENU1$
  1 MENCOM OVER CHDEP !
      13 = IF 1- MENVER 2 22 EFFLS 2 AFFLG= 13 =
      IF
 \ on vient de valider un choix vertical donc saut à la
 \ routine correspondante dans la matrice 10x6 et vive la
 \ multi-vectorisation !
     CHDEP @ 1- 10 * + 1- CHOIXACT
    ELSE DROP THEN FALSE
 \ sortie d'un choix vertical par ESC: On remonte au
 \ choix horizontal sans rien faire.
  ELSE TRUE \ sortie par ESC on sort de MENU
  THEN UNTIL DROP  ;
\ ---------------------------------------------------------
\ Exemple 4
\ Un autre possibilité de ces menus déroulants est de les
\ utiliser pour choisir parmi un groupe restreint de
\ données que l'on ne connait pas à la compilation du
\ programme mais stockées dans un fichier annexe. Je
\ m'explique: Vous avez à choisir parmi des groupes de 30
\ personnes  dont les noms, prénoms et âge sont dans des
\ fichiers ASCII sur disque de la forme:
\    \ Nom1              :Prénom1           :15 :12 :1 :73
```

```
\    \ Nom2               :Prénom2              :29
\    \ Nom3
\    \ Nom4               :Prénom4              :0   :10 :5 :88
\    \ etc dont on ne sait pas forcément le nombre total mais
\ dont on sait un maximum jamais atteint. Les NOMS vont
\ servir d'intitulés de menus et les prénoms vont servir de
\ messages correspondants. Las âges et dates de naissance
\ vont être mises dans un tableau annexe.
\ Le menu va être horizontal car sinon 30 lignes ne tiennent
\ pas à l'écran. La longueur de chaque menu doit être un
\ diviseur de 80 car sinon des noms seraient coupés par les
\ bords de l'écran. Dans l'exemple, il est de 20 et quatre
\ noms peuvent tenir dans une ligne. La notion d'horizontal
\ devient alors ambigüe car l'écran a l'aspect suivant: NOM1
\ NOM2 NOM3 NOM4
                                              NOM5 NOM6 etc...
\
\ Il faut donc un utilitaire permettant de lire ce fichier
\ et de renseigner le menu correspondant. C'est le mot  INIT
\ Mais auparavant quelques définitions
5 VARIABLES PARAM CPTR2 CPTR HANDLE2 NBPAR
\ NBPAR va contenir le nombre de paramètres derrière le
\ prénom
\ PARAM contient la zone où seront rangés ces paramètres
30 5  :DEFMEN NOMS$
\ C'est tout puisqu'on ne connait pas le contenu
: -TRAIL- -TRAILING BL SKIP ;  ( adr,len ---- adr1, len1 )
: DVAL -TRAIL- PAD PLACE BL PAD COUNT + C! PAD NUMBER?
       0= IF 2DROP 0 0 THEN DROP ;
\ ouverture pour séquentiel
   ( Adr avec 0 au bout,len ----- handle )
: OPENS
  2DUP DROP ?OPEN 0 (OPEN) ?DOS-ERR HANDLE !
  LINE# OFF PATHNAME! ;
12 STRING FICHNOMS$ 0 C, " FICHNOMP.TXT" FICHNOMS$ $!
\ Le 0 au bout sert pour la compatibilité avec le DOS où les
\ noms de fichiers doivent se terminer par un 0
\ ***** lecture séquentielle du fichier  *************
: INIT
   NBPAR ! PARAM ! HANDLE @ HANDLE2 ! OPENS  CPTR OFF
   BEGIN
     BUFFER HANDLE @ (GETLINE) NBRM @ CPTR @ = OR
     BUFFER COUNT -TRAIL- NIP 0= OR NOT
   WHILE
     BUFFER COUNT 2DUP ASCII : SCAN >R DUP >R NIP OVER -
     -TRAIL- LONGM @ MIN ADRM @ CPTR @ LONGM @ * + SWAP
     OVER LONGM @ BLANK CMOVE
     R> 1+ R> DUP 0<>
     IF 1- 2DUP ASCII : SCAN >R DUP >R
       NIP OVER -
       -TRAIL- HERE ADRMESS @ CPTR @ 2* + ! HERE PLACE
       HERE DUP C@ 20 MIN DUP 1+ ALLOT SWAP C! CPTR2 OFF
       BEGIN
         R> 1+ R> DUP 0<>
       WHILE 1- 2DUP ASCII : SCAN >R
         DUP >R NIP OVER -
         DVAL CPTR @ NBPAR @ * CPTR2 @ NBPAR @ 1- MIN +
         PARAM @ + C! CPTR2 1+!
       REPEAT
      THEN 2DROP
      CPTR 1+!
    REPEAT CPTR @ ADRM @ 3 - C! CLOSE
    HANDLE2 @ HANDLE ! ;
: INITNOM NOMS$ FICHNOMS$ PAD 5 INIT ;
: CHOIXNOM
   INITVID INITNOM 3 22 EFFLS 14 3 AT ." Quel nom ?
          1 15 5 MAT$  1 MENCOM  3 22 EFFLS ;
\ CHOIXNOM retourne le code et le numéro du choix
\ -----------------------------------------------------------
\ Exemple 5 Choix d'un fichier sur disquette pour par
\ exemple le lister
50 CONSTANT MAXDIR 16 STRING B$ VARIABLE CPTR
MAXDIR 16 :DEFMEN DIR$
14 STRING FICH$ 0 C, "                 " FICH$ $!
: INITDIR DARK CPTR OFF DIR$ MAXDIR NBRM ! ." Masque ? : "
  B$ INPUT$
  B$ NIP 0= B$ + 1- C@ ASCII : = OR
  IF " *.*" B$ APPEND$ THEN
  128 SET-DMA B$ PAD PLACE 0 PAD COUNT + C! PAD 1+
  16 (SEARCH0)
  IF
    BEGIN B$ DROP 1+ C@ ASCII : =
      IF B$ DROP 2 ELSE B$ DROP 0 THEN
      FICH$ $!  DMA 30 + DUP 12 0 SCAN DROP OVER -
      FICH$ APPEND$
      FICH$ LONGM @ MIN ADRM @ CPTR @ LONGM @ * +
      SWAP OVER LONGM @
      BLANK CMOVE  CPTR 1+! (SEARCH) NOT CPTR @
      MAXDIR 1- = OR
  UNTIL
  THEN CPTR @ ADRM @ 3 - C! ;
\ INITDIR commence par lire le directory dont le masque a
\ été rentré  à la  demande; Il peut comprendre le numéro
\ du drive, des jokers etc...
: CHOIX
  INITVID INITDIR DIR$ NBRM @ 0=
  IF  CR ." Aucun fichier sélectionné"
  ELSE 1 0 10 DIR$ 1 MENCOM 13 =
    IF  AFF1' -TRAILING FICH$ $! 0 FICH$ + C!
       HANDLE @ ECHO @ ECHO ON DARK
       FICH$ OPENS
       BEGIN GETLINE STOP? EOF? @ OR  UNTIL
       CLOSE ECHO ! HANDLE !
    ELSE DROP
    THEN
  THEN ;
\ Allez bon courage pour décortiquer tout ça. Mais ça
\ fonctionne ! Vous pouvez me croire. Avec ceci et pas mal
\ d'autres choses je gère l'ensemble des résultats
\ scolaires des élèves d'un établissement. (relevés
\ mensuels, trimestriels, moyennes, editions multiples,
\ histogrammes etc...)
\ Ca fonctionne sinon je me  serais fait tapé sur les
\ doigts.
\ Et tout en Forth avec des soupçons de LM
\ Si ça interesse quelqu'un !
```

Télématique

FORUM SAM*JEDI

Quelques mots sur le serveur SAM*JEDI: le nombre de communications est devenu consistant (moyenne de 3 heures de connexions par jour en semaine). Mais apparement, peu d'adhérents se connectent.

Le FORUM de SAM*JEDI est la "Hot-Line" télématique vous permettant de poser TOUTES les questions concernant le langage FORTH et les difficultés de programmation que vous pourriez avoir en utilisant TURBO-Forth. Nous garantissons (pour les questions concernant FORTH) une réponse dans un délai de huit jours maximum (sauf période de vacances...).

Maintenant, pour écrire au SECRETAIRE, il n'est plus nécessaire de chercher mon pseudo dans l'annuaire, mais simplement de déposer le message au SYSOP JEDI.

Dans le téléchargement de programmes, vous y trouverez tous les listings accompagnant les articles de la revue lorsque ceux-ci sont indiqués en tête d'article par "disponibilité:". La rubrique FORTH a été éclatée en six sous-rubriques:

1 Ctrl graphique et vidéo
2 Gestion de fichiers
3 Mathématiques
4 Turbo-Forth et extensions
5 Divers
6 Novix NC4016 et Harris RTX2000

soit un total de 49 fichiers disponibles au 19/03/89.
Sont également disponibles des programmes en C, dBASE, assembleur MSDOS, PROLOG et divers (code machine, BASIC, etc...).

SECRETAIRE   23.12.88 13h51

DISPONIBILITE DU MODULE 6: Enfin nous sortons le module 6, l'un des plus gros diffuses a ce jour, car il represente pres de 200ko de sources et de codes:

- FWIND et FWMENU, fenetrage en HERCUL.
- Tout WORDPERFORTH, un super (mais alors vraiment S-U-P-E-R) editeur ASCII pour remplacer EDIT.COM et EDITERR.MSG du module 1. WPF est un clone de WORDPERFECT qui tient, tenez-vous bien, en 16 Ko de code compile. Et tout ca, livre avec les fichiers source!! Non? SI!
- BUFFERS gestion de tampons
- LODBIN et SAVBIN.BLK pour F83 Laxen et Perry MSDOS, chargement et sauvegarde de code pre-compile.

Le module 6 est envoye contre 10 timbres a 3,70 Fr.
Commande a adresser a:
ASSOCIATION JEDI 17, rue de la Lancette 75012 PARIS
Si vous ne disposez pas de TURBO-Forth, vous pouvez acquerir tous les modules:
- modules M1 a M6: 190 Fr, port compris.
Sinceres salutations.

SECRETAIRE 23.12.88 14h04
REPONSE A F32 CONCERNANT DECOMPILATEUR:
Le mot SEE version MSDOS ou F83 CP/M fonctionne de la meme maniere. Chaque type de mot est defini par classe:
- constantes, variables, literal number, chaines caracteres...
Pour TURBO-Forth, sont egalement definis
- variables chaines, constantes et var 32 bits.
Un mot FORTH est decoupe en quatre zones
- nfa, lfa, cfa, pfa (zone parametre)
Pour un mot compile par : la zone parametre contient le code compile. A chaque paire d'octet de cette zone correspond un cfa qui est celui du mot a executer. Le pfa d'un mot est calcule par:

```
  ' mot 2+
```

Exemple:

```
  : AU-CARRE ( n --- n*n)
   DUP * ;
  ' AU-CARRE 2+  pointe sur l'adresse memoire contenant le cfa de DUP
  ' AU-CARRE 4+  pointe sur l'adresse memoire contenant *
```

Un nombre entier non defini comme constante est precede du code (LIT). Ex:

```
  : +32 ( n --- n+32)  32 + ;
  ' +32 2+ pointe sur l'adresse contenant le cfa de (LIT)
  ' +32 4+ pointe sur l'adresse contenant une valeur entiere 16 bits.
```

D'ailleurs, SEE +32 affiche

```
  : +32  (LIT) 32 ...etc...
```

Une chaine de caractere definie par " ou ." est precedee en version compilee par (") ou (.") et de la longueur de la chaine sur 8 bits, puis du contenu de la chaine.
Pour un mot defini autrement que par : il faudra tenir compte de la nature des donnes placees dans la zone parametrique. Attention, une definition du type

```
  CREATE TRUC 100 ALLOT
```

donne en decompilation:

```
  VARIABLE TRUC contenu nnnn
```

On ne peut mettre en evidence le mode de definition qu'en se rapportant rigoureusement a un type de definisseur precis: CONSTANT, VARIABLE, 2CONSTANT, 2VARIABLE, DEFER, VOCABULARY, MARK, et egalement STRING pour TF83.

Il est tres difficile de creer un decompilateur capable de tenir compte de tous les cas de figure: ceux existants deja ou ceux definis par l'utilisateur. Un exemple de piege: en TF83, la structure de type:

```
   : mot.... ASM[ ... ]FORTH... ;
```

n'a pas ete prevue initialement dans le decompilateur, donc SEE plantera quand il visualisera MOT. Ceci est du au fait que ASM[cree un pseudo-cfa et que ce pseudo-cfa ne pointe sur aucun en-tete.

Un autre piege tres difficile a eviter est celui des types de branchements. Par exemple, IF..THEN compile un ?BRANCH et SEE ne peut savoir ensuite si ce ?BRANCH vient d'un IF ou d'un CASE! Il en est de meme pour BEGIN..UNTIL. Certes, LE FORTH de GUILLAUMAUD arrive a resoudre ce probleme, mais sa resolution alourdit considerablement la structure du code FORTH (marqueur de type de branchement ou gestion de tableau)

En fait, les seuls decompilateurs exploitables et satisfaisants sont soit SEE (equipant d'uffice F83 et TF83) soit un decompilateur recursif permettant la mise en evidence du chainage des procedures.
Voila, j'espere avoir repondu sommairement a votre attente.

RHEMY 23.12.88 15h53
JE SUIS ARRIVE A TELECHARGER MAIS LES FICHIERS .EXE NE MARCHENT PAS.

SECRETAIRE 26.12.88 09h11
REPONSE A RHEMY: ATTENTION, LES FICHIERS QUI NE SONT PAS EN ASCII DOIVENT O-BLI-GA-TOI-RE-MENT ETRE CHARGES AVEC LE PROGRAMME TELECHAR LEQUEL FIGURE SUR LA DISQUETTE KERMIT ET FOURNIE PAR NOUS. POUR VOUS PROCURER CE PROGRAMME, LAISSEZ SIMPLEMENT VOS COORDONNEES DANS LA BAL SECRETAIRE ou au SYSOP SUR CE SERVEUR.
A L'ATTENTION DE TOUS: DORENAVANT, JEDI DIFFUSE GRATUITEMENT KERMIT, Y COMPRIS DISQUETTE ET FRAIS DE PORT. POUR VOUS LA PROCURER, LAISSEZ VOS COORDONNEES DANS LA BAL SECRETAIRE. MERCI.

SECRETAIRE 26.12.88 15h30
ENYGME DE SYRACUSE POUR MATHEUX:

```
\ Algorythme de la suite de Syracuse:
\ n pair? n2=n1/2   n impair? n2=n1*3+1
: SYRACUSE ( n1 --- n2 )
 DUP 2 MOD  IF 3 * 1+ ELSE  2/ THEN ;
: SUITE ( n ---)
  BEGIN  SYRACUSE  DUP .
  DUP 1 =   UNTIL ;
```

Le principe est simple, pour chaque valeur de n, SUITE delivre une sequence de nombres. Cette suite tend-elle a croitre ou a decroitre? A ce jour, aucune demonstration formelle ne peut y repondre. Et vous, qu'en pensez-vous? Essayez avec tous les nombres de 1 a 40: pour 27, le resultat est surprenant!

SECRETAIRE 30.12.88 13h23
```` M A I T E N A N C E S E R V E U R ````
Des travaux de maintenance vont un peu perturber le serveur. Le materiel sera bientot demenage et la ligne TRANSPAC transferee. Rien ne changera pour l'acces par 3615 a part qques perturbations dans les jours qui viennent.
Les nouveaux softs dispos en telechargement ont ete montes sur le nouveau serveur et seront disponibles des l'activation du nouveau chemin d'acces par TRANSPAC.

SECRETAIRE 05.01.89 13h21
NOUVEAUX SOFTS EN TELECHARGEMENT: De nouveaux softs ont ete places en telechargement:
- C: 4 logiciels dont le source et la doc de KERMIT PC (en anglais); tours de HANOI; jeu de la vie.
- FORTH, sous groupe base de donnes, programme RECORD.FTH (diffuse egalement sur module M6); sous groupe divers, QFILES.FTH (diffuse egalement sur M6) et FILTER.FTH (diffuse prochainement sur M7 et prochain JEDI no 49)

SECRETAIRE 11.01.89 13h52
UN META-COMPILATEUR 6809 SUR PC EN FORTH. J'ai mis au point un meta-compilateur generant du code 6809 sur PC et ecrit en TURBO-Forth. Certes, pratiquement plus personne ne s'interresse a cette gamme de materiel.
Mais cette experience a permis de mettre a jour les difficultes de developpement d'un generateur de code cible pour un processeur different de celui du pc.
Actuellement, je genere de petits programmes de tests (30 a 50 octets) qui tournent ensuite sur THOMSON TO7-70
Avec la compilation conditionnelle (voir ?\ et EXIST? sur un precedent message) on pourra creer des programmes uniques tournant sur différents types de systèmes equipés de processeurs très divers.

ESTHETE 14.01.89 18h07
RECTIFICATIF A MON PRECEDENT MESSAGE: QUE CONTIENT LE VFA? PLUS PRECISEMENT LE No DE FICHIER? MERCI.

SECRETAIRE 16.01.89 10h06
TURBO-FORTH SUR ATARI: Oui, notre TURBO-Forth en disquette 3'1/2 tourne sur ATARI sous emulateur MSDOS. Le test de la boucle a vide 10000 0 DO LOOP s'execute en environ deux secondes, ce qui est honorable pour un emulateur sur ce type de systeme. L'appel de l'editeur et le retour sous TURBO se passe sans probleme.
J'encourage donc tous ceux qui utilisent un ATARI a commander TF83 MSDOS s'ils disposent de cet emulateur et nous aident a developper des programmes en FORTH.

SECRETAIRE 16.01.89 10h59
LES DIFFERENTS CHAMPS EN FORTH sont:
- vfa: champ de vue
- lfa: champ de lien
- nfa: champ du nom
- cfa: champ de code d'execution
- pfa: champ parametrique

et dont les significations sont:
VFA (View Field Addres); est obtenu par >VIEW (cfa --- vfa); ce champ pointe sur le numero de fichier (handle) et le numero de bloc contenant la definition du mot compile. En F83 Laxen et Perry (MSDOS ou CP/M), un VIEW <mot> ouvre le fichier et liste le bloc contenant <mot>. En TURBO-Forth, ce champ a ete reaffecte au mot HELP et pointe sur un des fichiers d'autodocumentation.

LFA (Link Field Addres); est obtenu par >LINK (cfa --- lfa); ce champ pointe sur le nfa du precedent mot defini dans le vocabulaire.

NFA (Name Field Addres); est obtenu par >NAME (cfa --- nfa); ce champ 8 bits contient sur les cinq octets de poids faible, la longueur du libelle du mot, un bit de
````

validation, un bit de precedence (mot immediat ou non) et le bit de poids fort a 1.

CFA (Code Field Addres); est obtenu par ' <mot>; ce champ 16 bits contient la partie execution du mot de definition:
DOCOL pour un mot :
DOCON pour un mot CONSTANT
DOVAR pour un mot VARIABLE
DODEFER pour un DEFER
etc...
partie DOES> pour un mot defini par
un mot de type : <mot> CREATE ... DOES>
... ;
cfa+2 pour un mot en code machine.
Le cfa est le champ les plus important pour l'interpreteur-compilateur FORTH. Ce sont ces adresses qui sont compilees dans le pfa d'une definition : (deux-points).

PFA (Parameter Field Addres); est obtenu par >BODY (cfa --- pfa); ce champ est de longueur variable et contient selon le mot de definition:
- une succession de cfa si definition compilee par : (deux-points)
- valeur 16 bits si definition CONSTANT, VARIABLE, DEFER
- code machine si definition CODE
Si le mot de creation est CREATE, le champ pfa est vide au depart. Sa taille dependra des directives qui suivent la definition du mot:
CREATE 1BIT 4 C, reserve 1 octet dans le pfa de BIT
CREATE 1KBIT 1024 ALLOT reserve 1 Koctet dans le pfa de 1KBIT.
Un mot defini par CREATE sera execute comme un mot defini par VARIABLE.

DANELUZZO 16.01.89 16h49
JE REALISE UN PROTOTYPE DE MACHINE DE CONTROLE DE PIECES DE VISSERIE PAR VISION ROBOTIQUE POUR UN INDUSTRIEL. LE DEVELOPPEMENT DE F32 M'APPARAIT SEDUISANT PAR RAPPORT A LA RECONNAISSANCE DE FORMES. Y A T-IL UN MOYEN D'AVOIR UN ETAT DES TRAVAUX ET QUELS SONT LES CONTACTS POUR INTERVENIR?

SECRETAIRE 16.01.89 17h23
REPONSE A DANELUZZO: F32 AVANCE SON PETIT BONHOMME DE CHEMIN. EN CE QUI CONCERNE LE HARD,CONTACTER P. ORTAIS AU 1-69281090 HDB, OU LAISSER MSGS DANS BAL FF32 (LA BAL F32 AYANT UN PEU CAFOUILLE AU MOMENT DU CHGT DE MATERIEL ET TRANSFERT DE FICHIERS DU SERVEUR).

JLUC 18.01.89 16h26
SUITE A L'IDEE DE FORTH7
```
: QUERY2 250 TIB ! TIB 10 BDOS DROP
        TIB 1+ C@ 2+ DUP SPAN !
           #TIB ! 2 >IN ! CR ;
```
EN REMPLACANT QUERY PAR QUERY2 DANS LE MOT QUIT ET POUR PEU QUE DOSEDIT SOIT CHARGE ON OBTIENT UN EDITEUR DE COMMANDE. LE PROBLEME EST QUEL'ON A PLUS LES TOUCHES DE FONCTION !!!

FLIPO 18.01.89 19h43
Je me permets de temperer l'optimisme de notre Secretaire, quant a l'utilisation de TURBO-FORTH sur ATARI-ST :
1) Il faut acheter(ou pirater!)PC-DITTO
2) On perd tout le benefice d'avoir un processeur rapide (le 68000)...
3) On ne dispose plus d'une interface GEM. Developper sur ATARI sans GEM n'a pas de sens...
LA SEULE BONNE SOLUTION POUR UTILISER TURBO-FORTH SUR ATARI serait de
a) reecrire le code machine pour 68000
b) de mettre le SUPER-editeur sous GEM
c) d'ajouter une interface GEM, les appels au BIOS, le multi-taches etc..
Ce serait un projet interessant, mais compte tenu du peu de programmeurs en FORTH sur ATARI, le jeu en vaut-il la chandelle ? Il faut acheter (ou pirater!) PC-DITTO. De toute maniere, je ne suis pas pret a me lancer seul... S'il y a des gens interesses qu'ils se fassent connaitre. Je suis d'accord pour "collaborer"! On ne dispose plus d'une interface Pour mes besoins personnels, le Volks-FORTH egalement diffuse par l'association me convient parfaitement: il est rapide, completement interface avec GEM et permet tous les appels systemes.. Amities aux ATARIstes... et aux autres!

LAMBERTPH 20.01.89 13h03
IL ME SEMBLE QUE TIB PARTAGE L'ESPACE DE LA PILE RETOUR DANS TURBO-FORTH 256 OCTETS POUR LES DEUX N'EST CE PAS DANGEREUX? - DEMANDE CONFIRMATION OU EXPLICATION - MERCI - .

LEMAIGRE 20.01.89 15h28
QUI DISPOSERAI D'UN NOYAU 6809 OU 68HC11 EN FORTH79 OU 83 POUR METACOMPILE. MERCI.

FORTH7 23.01.89 14h16
CONCERNANT LE TIB (Tampon d'entree): La remarque de M. LAMBERTPH est tres judicieuse: il est exact que le TIB et la pile de retour progressent l'un vers l'autre sur les memes 256 octets selon une vieille habitude Forth heritee des systemes sur 8K ram.... Si les risques d'ecrasement sont faibles, ils existent et il serait bon de les separer dans les futures versions d'autant que le tampon du handle 0 (clavier justement) est inutilise jusqu'a present.
Editer les commandes du TIB via DOSEDIT comme le propose JLUC supprime en effet l'action des touches de fonction ou de controle mais repond a un besoin d'ergonomie legitime aujourd'hui. Je propose une autre solution: ajouter en forth de nouvelles touches d'edition en sus des Ctrl-X, BS, CR trop rustiques! La technique est simple: ainsi la ligne
```
' DARK CC @ CONTROL L 2* + !
```
permet d'effacer l'ecran avec Ctrl-L. Voici un autre exemple assez primitif:
```
: R-IN
  ?DUP 0= IF #TIB @ 2DUP TYPE THEN ;
   ' R-IN CC @ CONTROL R 2* + !
```
la touche Ctrl-R permet de rappeler la derniere commande entree. Avec un peu de forth, on pourrait faire mieux que DOSEDIT: suffit de demander...

JLUC 29.01.89 20h04
Suite a des messages dans le forum entre moi et FORTH7. Je viens de finir une version de DOSEDIT sans DOSEDIT externe, entierement en forth. Elle est totalement compatible avec les touches de fonction dont elle est une extension d'ailleurs. Toutes les entrees passant par le mot EXPECT beneficient de cet avantage (?): edition facile a l'aide des touches flechees, insertion et suppression. Mais comme ca prend de la place, a bientot de me lire peut-etre dans JEDI.

LEMAIGRE 30.01.89 19h28
A T'IL DEJA ETE TENTE DE FAIRE DIFFUSER TURBO FORTH GRACIEUSEMENT PAR UN IMPORTATEUR DE COMPATIBLE A LA VENTE DE LA MACHINE. LA DOC POUVANT ETRE OBTENUE AUPRES DE JEDI, BIEN SUR. UN BON PROGRAMME DE DEMO DE PERFORMANCE SERAIT NECESSAIRE. SLT

LEMAIGRE 30.01.89 19h28
UNE SOCIETE ALLEMANDE DIFFUSE UNE VERSION MASQUEE DU 68HC11 MOTOROLA AINSI QU'UNE CARTE D'EVALUATION POUVANT SERVIR DE SYSTEME DE DEVELOPPEMENT ET UN ASSEMBLEUR SPECIFIQUE (FORTH BASED). J'ATTEND LEUR DOC SI INTERET CONTACTEZ MOI.

SECRETAIRE 31.01.89 11h50
EMPILAGE ET DEPILAGE 32 BITS SUR PILE RETOUR:
Vous pouvez definir : 2>R >R >R ; et
: 2R> R> R> ; mais en code machine c'est mieux:
```
 CODE 2R>
  0 [RP] DX MOV  RP INC  RP INC
  0 [RP]  AX MOV  RP INC  RP INC
  2PUSH END-CODE
 CODE 2>R
  AX POP  DX POP  RP DEC  RP DEC
  AX 0 [RP] MOV  RP DEC  RP DEC
  DX 0 [RP] MOV  NEXT END-CODE
FONCTIONS LOGIQUES 32 BITS:
 CODE 2AND ( d1 d2 --- d3)
  AX POP  DX POP  BX POP  BX AX AND
  BX POP  BX DX AND  2PUSH END-CODE
 CODE 2OR  ( d1 d2 --- d3)
  AX POP  DX POP  BX POP  BX AX OR
  BX POP  BX DX OR  2PUSH END-CODE
 CODE 2XOR ( d1 d2 --- d3)
  AX POP  DX POP  BX POP  BX AX XOR
  BX POP  BX DX AND  2PUSH END-CODE
 CODE 2NOT ( D --- D')
  AX POP  AX NOT  DX POP  DX NOT
  2PUSH END-CODE
FLAGS BOOLEENS 32 BITS:
 CODE 2TRUE ( --- dtf)
  65535 # DX MOV  DX AX MOV
  2PUSH END-CODE
 CODE 2FALSE ( --- dff)
  0 # DX MOV  DX AX MOV  2PUSH END-CODE
```

SECRETAIRE 31.01.89 12h05

```
CODE UD< ( d --- f)
  DX POP  CX POP  BX POP  AX POP
  AX CX CMP U<
  IF   0 # AX MOV
  ELSE BX DX CMP  U<
     IF   0 # AX MOV
     ELSE 65535 # AX MOV THEN
  THEN 1PUSH END-CODE
```

SECRETAIRE 31.01.89 12h09
REPONSE A LEMAIGRE: ADRESSE ANGELIKA FLESH FORTH SYSTEME:
FORTH SYSTEME (ANGELIKA FLESH)
KUHNHEIMERSTRASSE 21
POSTFACH 1103
D - 7814 BREISACH (RFA/BRD)
TEL:19-49 7667551
FAX:19-49 7667555
MESSAGERIE PAR MODEM,
PROTOCOLE 8N1, 300, 1200, 2400 BDS AU 19-49 7667556
DIFFUSE PRODUITS UR/FORTH, HARRIS RTX2000, FORTH LMI AVEC METACOMPILATEURS 8086/8088, Z80, 8080, 68000, 6809, 6502, 68HC11, etc...

SECRETAIRE 01.02.89 12h11
NOUVEAU SOFTS EN TELECHARGEMENT:
EN dBASE: SAUVEGARDE ET RESTAURATION VIDEO AVEC SOURCE ET DOC
EN C: GESTION CALENDAIRE ET UN UTILITAIRE CONVERSION .COM EN .HEX INTEL.
EN DIVERS: GESTION TAMPON COMMANDES DOS ET GENERATEUR .COM A PARTIR IMAGE VIDEO TEXTE OU GRAPHIQUE CGA, AVEC DOC.
EN FORTH, RUBRIQUE 4, METACOMPILATEUR CIBLE 6809. CE PROGRAMME EST ENCORE UN PROTO. POUR LE MOMENT, LES TESTS CONFIRMENT BIEN QUE C'EST DU CODE 6809 QUI EST GENERE.
CE PROGRAMME EST UN EXEMPLE D'APPLICATION DE METAGENERATION FORTH PC VERS CIBLE NON PC. IL PEUT ETRE ADAPTE SANS DIFFICULTE A D'AUTRES PROCESSEURS. POUR CELA, IL SUFFIT DE DEFINIR UN AUTRE ASSEMBLEUR. J'AI ESSAYE AUSSI AVEC L'ASSEMBLEUR 8080 INTEL DE F83 CP/M MAIS JE N'AI PAS ENCORE FAIT LES TESTS.
CE PROGRAMME REPONDRA CERTAINEMENT AUX PREOCCUPATIONS DE LEMAIGRE S'IL A LA PATIENCE DE L'ADAPTER AU 8051 RTC.
A TERME, NOUS SOUHAITONS FOURNIR DES METAGENERATEURS DE CODE POUR LA PLUPART DES PROCESSEURS EXISTANTS (80386, 68000, HARRIS RTX 2000, ...). MAIS COMME ON NE PEUT TOUT FAIRE TOUT SEUL, J'ATTEND BEAUCOUP DE VOS COGITATIONS.

SECRETAIRE 03.02.89 13h27
JOURNEES FORTH 1989: ORGANISEES PAR FIG HAMBOURG (RFA), LIEU AIX LA CHAPELLE (AACHEN) LES 7, 8 ET
9 AVRIL 1989. PRIX: PARTICIPANT 100 DM
MEMBRE FIG HAMBOURG 60 DM
ETUDIANT 30 DM
SOCIETE 120 DM
STAND 100 DM PAR M2
4 REPAS 66 DM
CONTACT: RLOF KRETZSCHMAR 18H A 20H
AU 19-49 2401.4390 (RFA)
OU ECRIRE:
FORTH - Tagungsburo 89
c/o Rolf KRETZSCHMAR
Rote Gasse 7
D - 5112 BAESWEILER

SECRETAIRE 03.02.89 13h35
POUR FAIRE DIRE OUI AU LIEU DE OK A FORTH (F83 L&P ET TF83)

```
 : QUITTE
  SP0 @ 'TIB ! [COMPILE] [
  BEGIN RP0 @ RP! STATUS QUERY RUN
  STATE @ NOT IF ." Oui" THEN  AGAIN ;
```

ET TAPPEZ QUITTE, MAINTENANT " Oui" S'AFFICHE A LA PLACE DE " OK".

SECRETAIRE 03.02.89 13h38
ET SI TOUT NOMBRE TAPE ETAIT SYSTEMATIQUEMENT AU FORMAT 32 BITS?? FASTOCHE:

```
: NOUVEAU-INTERPRET
  BEGIN ?STACK DEFINED
  IF EXECUTE  ELSE NUMBER THEN FALSE DONE?  UNTIL ;
: NOUVEAU-]  STATE ON BEGIN ?STACK DEFINED DUP
  IF 0> IF EXECUTE ELSE , THEN
  ELSE DROP NUMBER DOUBLE?
  IF [COMPILE] LITERAL [COMPILE] LITERAL
  ELSE [COMPILE] LITERAL THEN
  THEN THEN TRUE DONE? ;
: NOUVEAU-RUN  STATE @ IF NOUVEAU-]
  STATE @ NOT IF NOUVEAU-INTERPRET THEN
  ELSE NOUVEAU-INTERPRET THEN ;
```

REMPRENDRE LA DEFINITION DE QUITTE (MESSAGE PRECEDENT) ET REMPLACER RUN PAR NOUVEAU-RUN. ENSUITE TAPEZ QUITTE ET A PARTIR DE MAINTENANT FORTH DEPOSE SYSTEMATIQUEMENT UN NOMBRE 32 BITS SUR LA PILE:
16 3. D+ D. AFFICHE 19
SI VOUS AVEZ BESOIN DE CONVERTIR UN NOMBRE 32 BITS EN 16 BITS:

```
 : D>S DROP ;
```

NE FONCTIONNE QUE POUR LES VALEURS POSITIVES INFERIEURES A 65536.

DURANTON 03.02.89 19h55
Aux specialistes de TURBO FORTH. Je n'arrive pas a faire fonctionner correctement le multi tache. Le petit programme d'essai suivant fonctionne sous Laxen et Perry mais pas sur turbo-forth.

```
BACKGROUND: TOTO 100 0 DO I .
PAUSE LOOP ;
TOTO WAKE MULTI
```

donne comme resultats :
1 2 3 4 5 6 7 8 9 1w 1w w 1w 1w etc...
Il semble que les nouvelles definitions de EMIT avec appel BIOS sont incompatibles avec le multitache. Si quelqu'un a la solution permettant de modifier ce comportement du multitache... Merci.

FORTH7 06.02.89 19h33
REPONSE A M. DURANTON SUR LE MULTITACHE.
Ni EMIT ni le bios ne sont en cause! Le mot . qui affiche un nombre utilise une zone PAD de conversion alphanumerique tache-dependante. Par defaut le systeme affecte a une nouvelle tache les memes variables USER que celles en a cours lors de la definition de la tache et si on n'y prend garde ca coince un max entre les taches... Une solution au probleme du PAD (qui est lie a DP) est donnee dans le fichier CLOCK.FTH du module 3 (horloge en multitache). Ne pas oublier encore: une tache doit se terminer par STOP ou ne pas se terminer du tout (BEGIN...AGAIN).
Le multitache? Mais c'est tres simple! N'est-ce pas MM. de MICROSOFT?

SECRETAIRE 08.02.89 12h11
DERNIERE NOUVELLE DU RTX 2000 HARRIS:
LES ESSAIS DU SYSTEME DE DEVELOPPEMENT HARRIS RTXDS EQUIPE DU RTX 2000 SONT ASSEZ FAVORABLES. LES BENCHMARKS REALISES PAR FF32 DONNENT UNE RAPIDITE DE CERTAINS PROGRAMMES ECRITS EN FORTH SUPERIEURE AUX PROGRAMMES EQUIVALENTS ECRITS EN C ET ASSM 68030 (30 MHZ), SOIT DES PERFORMANCES SUPERIEURES A UN MICROVAX 2.
NOUS AURONS D'AILLEURS L'OCCASION PROCHAINEMENT D'OUVRIR UNE RUBRIQUE MIXTE NOVIX-HARRIS POUR VOUS PERMETTRE DE TELECHARGER DES SOFTS POUR CES DEUX PROCESSEURS. NATURELLEMENT, TOUS CES DEVELOPPEMENTS PRIVILEGIERONT LES PROGRAMMES ASSOCIES A TURBO-FORTH PLUTOT QUE LE FORTH LMI.

SECRETAIRE 09.02.89 11h48
CHAINES DE CARACTERES AVEC CODES DE CTRL Dans un ancien message, j'expliquai comment concatener un code de CTRL dans une chaine de caratere. Mais comment faire s'il y en a plusieurs? Voici la reponse:

```
 : $CTRL: CREATE DOES> COUNT ;
```

Voici comment definir un CRLF contenant CTRL D CTRL A:

```
 $CTRL: CRLF$  HERE 0 C,
   CONTROL D C,  CONTROL A C,
   HERE OVER - 1- SWAP C!
```

Ceci est particulierement interressant pour controler une imprimante:

```
$CTRL: 12PITCH   HERE 0 C,
  27 C, ASCII ] C, ASCII 5 C, ASCII w C,
  HERE OVER - 1- SWAP C!
$CTRL: 17PITCH   HERE 0 C,
  27 C, ASCII ] C, ASCII 6 C, ASCII w C,
80 STRING TITRE$
12PITCH TITRE$ $!
" COMPTES 1988 " TITRE$ APPEND$
17PITCH TITRE$ APPEND$
" Edition du 09/02/89" TITRE$ APPEND$
CRLF$ TITRE$ APPEND$
PRINTING ON TITRE$ TYPE PRINTING OFF
```

Ainsi le controle d'impression est contenu dans la chaine TITRE$.

SECRETAIRE 09.02.89 12h18
ERREUR DANS CRLF$ DEFINI PRECEDEMMENT:
REMPLACER CONTROL D PAR 13
CONTROL A PAR 10

Appendix A: Forth Implementation of the Prolog Virtual Machine

Glossary

The Forth words used to simulate the Prolog machine (screens 29-35) are described below.

`CALL (arity ^atom - )` Takes an integer (arity) and a pointer to the pfa of a constant record and searches the procedure records for a procedure with the specified arity. Backtracks if unsuccessful. Otherwise, if there are multiple applicable clauses, backtracking information is saved: a pointer to the next clause record, a pointer to the next most recent choice-point frame in the control stack and the trail and global stack-pointers. Sets the execution mode to match. Also saves the top of the Forth return stack in the next Prolog control stack frame and initializes the argument pointer to the first argument position in the next control frame (the first memory location after the 12 bytes of control information).

`ENTER (- )` Commits to execution of the body of a clause. Sets execution mode to arg and adjusts control stack pointers.

`RETURN (- )` Indicates success exit. Gets the return pointer saved by `CALL` and pushes it on the Forth return stack. Sets execution mode to arg and adjusts control stack pointers, reclaiming control stack frames where possible.

`CONST (^atom - )` Match mode: tests whether the next argument is a constant whose name is given by atom. Backtracks if they don't match; continues with Forth execution otherwise. Arg mode: builds an argument with tag = 2 and val = ^atom.

`VAR (n - )` Match mode: de-references the variable specified by the input index and then, if the variable is unbound, binds it to the argument; otherwise `VAR` tests whether the argument matches the binding. Backtracks if they don't match; continues with Forth execution otherwise. Arg mode: copies the variable binding to the next argument position.

`FUNCTOR (arity ^atom - )` Match mode: tests whether the next argument is a structured term with the specified arity, whose functor is named by atom. Backtracks if they don't match; otherwise saves the argument pointer and resets it to point to the first argument of the structured term (i.e., to point to the first cell of the argument field of the structure record pointed to by the input argument). Forth execution continues from this point. Arg mode: allocates space for a structure record from the global stack and then sets the first two fields of the record (i.e., functor name and arity). Builds an argument with tag = 3 and val = `^structure`. Saves the argument pointer and resets it to point to the first cell of the argument field of the structure record. Forth execution continues from this point.

`POP (- )` Restores the argument pointer in both execution modes.

Appendix B: Prolog-PVM Compiler

```
/*  =====    =====    =====    =====    =====   */
% COMPILER

% All args are output
% NVars is the number of vars in the clause
% Code is the object code
% Sent is a list of tokens, terminated by the token '.'
% The variable Sent is bound by the procedure "read_in"

compile_clause(Pred/Arity,Nvars,Code) :-
    read_in(Sent),
    horn_clause(Pred/Arity,Vars,[ ],Cd,Sent,_),
    nrev(Cd,Code),memberchkN(-1,Vars,-1,Nvars).

/*  =====    =====    =====    =====    =====   */
% Old is the input code sequence ([ ] here)

horn_clause(Pred/Arity,Vars,Old,['RETURN' |New]) -->
    atmf(Vars,Old,[_,_,_|N1],Pred/Arity),
    (["."],{New=N1}|
    [":-"],atmfs(Vars,['ENTER' |N1],New ) ).

/*  =====    =====    =====    =====    =====   */

% atmfs are terms that aren't numbers or variables
%   Arg1: list of vars in the clause
%   Arg2: input code sequence
%   Arg3: output code sequence

atmfs(Vars,Old,New ) -->
    atmf(Vars,Old,N1,_),
    (['.'],{New=N1}|[','],atmfs(Vars,N1,New) ).

atmf(Vars,Old,['CALL',Pred,0|Old],Pred/0) -->
    atom_name(Pred).

atmf(Vars,Old,['CALL',Pred,Arity|N1],Pred/Arity) -->
    atom_name(Pred),['('],args(Vars,Old,N1,0,Arity),[')'].

/*  =====    =====    =====    =====    =====   */

% simple_term parses terms, lists and structured terms in
%       functional form
% This is basically a case analysis and is meant to be
%       deterministic, thus the cut
%   Arg1: Variable symbol list (difference list - built
%           during compilation)
%   Arg2: Input Code list (input)
%   Arg3: Output Code list (output)

simple_term(_,_,_) -->
['.'],!,fail.

simple_term(Vars,Old,['POP' |N1]) -->
    atom_name(Pred),['('],
    args(Vars,['FUNCTOR',Pred,Arity|Old],N1,0,Arity),[')'],!.

simple_term(Vars,Old,New ) -->
    variable(Vars,Old,New),!.

simple_term(_,Old,New ) -->
    constant(Old,New),!.

simple_term(Vars,Old,New ) -->
    ['('],(simple_term(Vars,Old,New ) ;
    conjunction(Vars,Old,New ) ),[')'],!.

simple_term(Vars,Old,New ) -->
    list(Vars,Old,New ),!.

conjunction(Vars,Old,New ) -->
    simple_term(Vars,Old,N1 ),[','],
    (simple_term(Vars,N1,New ) | conjunction(Vars,N1,New ) ).

/*  =====    =====    =====    =====    =====   */

% args parses arguments
%   Arg1: - Arg3: see arguments for simple_term
%   Arg4: arg count (input)
%   Arg5: arg count (output)

args(Vars,Old,New,Num0,Num1 ) -->
    simple_term(Vars,Old,N1),
    ([','],{Num is Num0+1},args(Vars,N1,New,Num,Num1) |
    {New = N1,Num1 is Num0+1}).

/*  =====    =====    =====    =====    =====   */

% char(N,Atom,M)   - M is the N-th character of Atom
% ascii(M,N)       - the ascii code for M is N

atom_name(Name) -->
    [Name],{char(1,Name,M),
    ascii(M,N),ascii(a,_a),ascii(z,_z),_a =< N,N =< _z},!.

variable(Vars,Old,['VAR',Num |Old]) -->
    [Var],{char(1,Var,L),ascii(L,M),
    (ascii('_',M) ;
    ascii('A',_a),ascii('Z',_z),_a =< M,M =< _z),
    memberchkN(Var,Vars,0,Num)},!.

constant(Old,[Inst,X |Old]) -->
    atom_name(X),{Inst = 'CONST'}|
    [X],{integer(X),Inst = 'INTEGER'}.

list(Vars,Old,New) -->
    ['[',']'],{New=['CONST',nil|Old]} |
    ['['],simple_term(Vars,['FUNCTOR',cons,2|Old],N1),
    list_tail(Vars,N1,New ) .

/*  =====    =====    =====    =====    =====   */
/* Auxiliary relations */

list_tail(Vars,Old,['POP','CONST',nil|Old]) -->
    [']'].

list_tail(Vars,Old,['POP' |New]) -->
    ['|'],simple_term(Vars,Old,New),[']'].

list_tail(Vars,Old,New) -->
    [','],
    simple_term(Vars,['FUNCTOR',cons,2|Old],N1) ,
    list_tail(Vars,N1,New) .

% memberchkN

memberchkN(X,[X|_],M,N) :- N is M+1,!.
memberchkN(X,[_|T],L,N) :- M is L+1,memberchkN(X,T,M,N).

% naive reverse

nrev([X|L0],L) :-
    nrev(L0,L1),concatenate(L1,[X],L).
nrev([ ],[ ]).

% concatenate

concatenate([ ],L,L).
concatenate([X|L1],L2,[X|L3]) :- concatenate(L1,L2,L3).
```

```
SCR# 2
\  Primitive Macro Facility                              LLO 6/16/86
\  At compile time, copy the code over in-line
\  May not work for control structures
\  RUN-SYMBOLS 16 + @ returns the cfa of EXIT
: ;M [COMPILE] ; IMMEDIATE
   DOES> BEGIN DUP @ DUP RUN-SYMBOLS 16 + @ = NOT
         WHILE , 2+
         REPEAT 2DROP ;  IMMEDIATE
: :M : ;
```

```
SCR# 3
\ Stack Definitions                                      LLO 6/16/86
\ ^ARG.STACK is used for argument pointers
\ pushed by FUNCTOR, popped by POP

CREATE STRUCTURE.STACK  6000 ALLOT      \ structure stack
CREATE CONTROL.STACK    6000 ALLOT      \ control stack
CREATE TRAIL            1000 ALLOT      \ trail stack
CREATE ^ARG.STACK       1000 ALLOT      \ arg pointer stack
```

```
SCR# 4
\ Prolog Machine Registers                               LLO 6/16/86
\ VALUE is a multiple cfa word
\ access value by name (ie. X vs. X @)
\ store value indicated by IS  ( ie. 2 IS X vs. 2 X !)

0 VALUE CF                              \ Current   Frame
0 VALUE NF                              \ Next      Frame
0 VALUE BF                              \ Backtrack Frame
0 VALUE SS                              \ Structure Stack
0 VALUE TS                              \ Trail  Stack
```

```
SCR# 5
\ Register to Register Operations                        LLO 6/16/86

: BF>CF   BF IS CF ;               \ on backtracking
: NF>CF   NF IS CF ;               \ on procedure call
: NF>BF   NF IS BF ;               \ on choice point call
: CF>NF   CF IS NF ;               \ on deterministic return
```

```
SCR# 6
\ Context Save Operations                                LLO 6/16/86
\ IP> and >IP correspond to R> and >R

:M RP>Stack ( -- )  IP> NF      ! ;M
:  BP>Stack ( n -- )    NF 2+   ! ;
:  CF>Stack ( -- )    CF NF 4  + ! ;
:  BF>Stack ( -- )    BF NF 6  + ! ;
:  SS>Stack ( -- )    SS NF 8  + ! ;
:  TS>Stack ( -- )    TS NF 10 + ! ;
```

```
SCR# 7
\ Context Restore Operations                             LLO 6/16/86
:M StackNF>RP ( -- ) IP> DROP NF @ >IP  ;M  \ ret from unit
:M Stack>RP   ( -- ) IP> DROP CF @ >IP  ;M
:  Stack>BP ( -- n)  CF 2+  @           ;
:  Stack>CF ( -- )   CF 4  + @ IS CF    ;
:  Stack>BF ( -- )   CF 6  + @ IS BF    ;
:  Stack>GS ( -- )   CF 8  + @ IS SS    ;
:  Stack>TS ( -- )   CF 10 + @ IS TS    ;
```

```
SCR# 8
\ Temporaries, Equates, Tags                             LLO 6/16/86
\ Temporary Variables
0 VALUE ARG                             \ Argument Pointer
0 VALUE NVARS                           \ cache
0 VALUE ARITY                           \ cache
  VARIABLE ARG.FLG                      \ arg-match mode flag
  VARIABLE COPY.FLG                     \ copy mode flag
\ Equates
 4  CONSTANT BYTES/CELL
12  CONSTANT BYTES/FRAME
\ Tags  -  Here the tag is the high order 16 bits
 1  CONSTANT VAR.TYPE
 2  CONSTANT CONST.TYPE
 3  CONSTANT FUNCT.TYPE
```

```
SCR# 9
\ Record Manipulation                                    LLO 6/16/86
\ Code records
: CODE>ARITY   ( ^code -- n )     2+  C@ ;
: CODE>NVARS   ( ^code -- n )     3 + C@ ;
: CODE>PROC    ( ^code -- addr ) 4 +     ;

\ Procedure Records
: PROC>CODE  ( ^clause -- ^code )  4 + @ ;   \ procedure > code
: PROC>ARITY ( ^clause -- n ) 2+ @ ;         \ procedure > arity
```

```
SCR# 10
\ Record Manipulation                                    LLO 7/20/86
\ Structure Records
: STRUC>FUNCTOR ( ^functor -- ^atom )    @ ;     \ func > name
: STRUC>ARITY   ( ^functor -- arity ) 2+ @ ;     \ func > arity
: STRUC>ARGS    ( ^functor -- ^args ) 4 +  ;     \ func > args

\ Term References
: >TYPE       ( term.ref -- type  )    @ ;    \ ref > type
: >POINT      ( term.ref -- ^term ) 2+ @ ;    \ ref > pointer
```

```
SCR# 11
\ Procedure Search                                       LLO 6/16/86
: FIND.PROC ( n pfa -- ^code | flag )
\ Find a procedure of given arity and functor
\ arity = n, functor = pfa
\ Return FALSE if not found
\ Return pointer to first code record if found
  BEGIN
  @ DUP                            \ get pointer to clause records
  IF 2DUP PROC>ARITY =             \ compare arity
     IF TRUE ELSE FALSE THEN       \ if = then we're done
  ELSE  TRUE THEN                  \ if link = 0 then we're done
  UNTIL SWAP DROP DUP              \ clean stack
  IF PROC>CODE TRUE THEN ;         \ convert to code record pointer
```

```
SCR# 12
\ Variable Manipulation                                  LLO 6/16/86
: CREATE.UNBOUND.VAR ( addr -- )
\ Create an unbound variable at addr
  VAR.TYPE OVER 2! ;      \ store
: INIT.VARS ( -- )
\ create unbound variables in the control frame
  NVARS ?DUP
  IF    NF BYTES/FRAME +
        ARITY BYTES/CELL * +    DUP ROT BYTES/CELL * + SWAP
        DO    I CREATE.UNBOUND.VAR    BYTES/CELL +LOOP   THEN ;
: RESET.VARS ( top.TS bottom.TS -- )
\ reset the variables on the trail stack
  2DUP =   IF 2DROP  ELSE DO I @ VAR.TYPE OVER 2! 2 +LOOP  THEN ;
```

```
SCR# 13
\ Saving State                                           LLO 6/16/86
:M  RESET.RP ( ^code -- )
\ Reset the return stack pointer
  IP> DROP CODE>PROC >IP ;M
: SAVE.BACKTRACK ( ^code -- ^code )
\ Save appropriate information at a backtrack point
  DUP @ DUP BP>Stack     \ get link to next code record
  IF BF>Stack            \ if a choice point, save old BF
     NF>BF               \ current frame is new backtrack frame
     SS>Stack TS>Stack   \ save stack pointers
  THEN ;
```

```
SCR# 14
\ Adjust Pointer to Next Frame                           LLO 6/16/86
: SET.NF ( ^code -- ^code )
\ reset NF register
  CODE>ARITY BYTES/CELL * BYTES/FRAME + CF + IS BF ;
: RESET.NF ( -- )
\ reset NF pointer
  NF BYTES/FRAME +
  ARITY NVARS + BYTES/CELL * +
  IS NF ;                         \ adjust next frame pointer
: INIT.^ARG ( -- )
\ init ARG register
  NF BYTES/FRAME + IS ARG ;
```

```
SCR# 15
\ General Backtracking                                   LLO 6/16/86
: BACKTRACK ( -- )
\ Restore context
\ BF>CF                           \ make the BF current
\ Stack>BP DUP RESET.RP           \ get pointer to code record
\ @ DUP BP>Stack 0=               \ get link to next code record
\ IF Stack>BF THEN                \ reset BF if this not choice pt
\ Stack>PF Stack>GS               \ restore PF and SS pointers
\ TS Stack>TS                     \ restore TS
\ TS RESET.VARS                   \ reset vars on the trail
\ INIT.^ARG
\ SET.NF                          \ reset NF
  ." not implemented " ;
```

```
SCR# 16
\ Backtracking on Unification Failure
: RETRY ( -- )
\ Restore context
  CF                            \ get a copy of CF
  BF>CF                         \ make the BF current
  Stack>BP DUP RESET.RP         \ get pointer to code record
  DUP CODE>NVARS IS NVARS
  @ DUP BP>Stack 0=             \ get link to next code record
  IF Stack>BF THEN              \ reset BF if this not choice pt
  Stack>GS                      \ restore SS pointer
  TS Stack>TS                   \ restore TS
  TS RESET.VARS                 \ reset vars on the trail
  INIT.VARS  INIT.^ARG
  IS CF   ;                     \ restore CF
```

```
SCR# 17
\ Binding Trail                                    ... 7/26/86
\ keep track of variable bindings that may need
\ to be reset on backtracking

: >TRAIL ( ^var -- )
\ put a variable cell address on the trail
  TS !  TS 2+ IS TS ;

: >TRAIL? ( ^var -- ^var )
\ trail a variable if necessary
  DUP BF U<  OVER SS U<  OR
  IF DUP >TRAIL THEN ;
```

```
SCR# 18
\ Creating and Fetching a Binding                  ... 7/26/86
: BIND ( ^term type ^var -- )
\ bind a variable
  >TRAIL? 2! ;          \ smash variable cell

: >ULT.BINDING ( ^term0 -- ^term1 )
\ completely dereference a variable binding
  BEGIN
  DUP >TYPE VAR.TYPE =                \ check for variable
   IF DUP >POINT OVER =               \ check if unbound var
     IF TRUE ELSE >POINT FALSE THEN   \ chase pointer if bound
   ELSE TRUE THEN                     \ not variable then done
  UNTIL ;
```

```
SCR# 19
\  Save and Restore Arguments                      LLO 6/16/86

: POP.ARG ( -- ^term )
\ pop an argument off control/structure stack
\ leave term pointer on data stack
  ARG DUP BYTES/CELL + IS ARG ;
: PUSH.ARG ( ^term type -- )
\ push an argument onto the control/structure stack
  ARG 2!  ARG BYTES/CELL + IS ARG ;
```

```
SCR# 20
\ Save and Restore Arg Pointers                    ... 7/26/86
: PUSH.^ARG ( addr -- )
\ save then reset argument pointer
  2 ^ARG.STACK  ARG OVER @ !  +!  IS ARG ;
: POP.^ARG ( -- )
\ restore argument pointer
  ^ARG.STACK -2 OVER +!   @ @ IS ARG  ;

: ARG&TYPE ( -- ^term type )
\ pop an argument, dereference it if necessary, return with type
  POP.ARG >ULT.BINDING DUP >TYPE ;
```

```
SCR# 21
\ Unify Variable (match mode)                      LLO 7/20/86
\ see VAR, screen 33
: UNIFY.VAR  ( ^var -- flag )
\ unify the variable with an argument
  ARG&TYPE  VAR.TYPE =           \ get the argument
  IF 2DUP U> IF SWAP THEN 2+ !  \ bind variables
  ELSE 2@ ROT BIND THEN  TRUE ; \ bind it to the variable
```

```
SCR# 22
\ Unify Constant                                   ... 7/22/86
\ see VAR, screen 33
: UNIFY.CONST ( ^const -- flag )
\ unify the constant with an argument
  ARG&TYPE  CASE                     \ get the argument
  VAR.TYPE
   OF >R 2@ R> BIND TRUE ENDOF
  CONST.TYPE
   OF >POINT SWAP >POINT =           \ compare pointers
       IF TRUE ELSE FALSE THEN       \ succeed if pointers match
  ENDOF
  SWAP FALSE ENDCASE   ;             \ nothing else recognized
```

```
SCR# 27
\ Copy a Structure (copy mode)                             LLO 7/20/86
\ see FUNCTOR, screen 35
: MATCH.VAR ( arity ^atom ^var -- flag )
\ Builds a structure record and binds it to the variable
\ The structure args are built by the remainder of the head code
  COPY.FLG ON                   \ copy variable from NF not CF
  SS FUNCT.TYPE ROT BIND        \ bind the variable to funct rec
  COPY.FUNCT    TRUE ;          \ set to copy functor, return T
```

```
SCR# 28
\ Prolog Virtual Machine Instructions
\ Note: VAR takes a byte offset from the start of a frame
\ (base address) to compute the address of the variable.
\ ARG.FLG = 0, COPY.FLG = 0 indicates match mode
\ ARG.FLG not 0, COPY.FLG = 0 indicates arg mode
\ ARG.FLG not 0, COPY.FLG not 0 indicates copy mode
\ in copy mode, nesting is handled by decrementing ARG.FLG
\ at the start of a structure, and incrementing on exit
\ (via POP).
\ COPY.FLG is used to indicate which frame pointer (CF or NF)
\ should be used for the base address of the variable.
```

```
SCR# 29
\ Prolog Machine Instruction     CALL                LLO 6/16/86
: CALL ( n cfa -- )
\ Call a Prolog procedure
  ARG.FLG OFF                   \ swith mode to "match"
  FIND.PROC                     \ get pointer to procedure rec
  IF DUP CODE>NVARS IS NVARS    \ cache NVARS
     DUP CODE>ARITY IS ARITY    \ cache ARITY
     INIT.VARS RP>Stack         \ init vars and set up stack
     SAVE.BACKTRACK
     INIT.^ARG                  \ set argument pointer
     4 + >IP EXIT               \ pass control to procedure
  ELSE BACKTRACK THEN   ;       \ backtrack if no procedure
```

```
SCR# 30
\ Prolog Machine Instruction     ENTER               LLO 6/16/86

: ENTER ( -- )
\ Enter a procedure and begin execution of the body
  ARG.FLG ON                    \ switch mode to "arg"
  COPY.FLG OFF
  CF>Stack                      \ adjust frame pointers
  NF>CF RESET.NF                \ adjust next frame pointer
  INIT.^ARG ;                   \ set arg pointer for next call
```

```
SCR# 23
\ Unify Structure                                        LLO 7/20/86
\ see VAR, screen 33

: UNIFY.FUNCT ( ^functor -- flag )
\ unify the functor with an argument
  ARG&TYPE  CASE                                  \ get the argument
  VAR.TYPE   OF >R 2@ R> BIND         TRUE ENDOF
  FUNCT.TYPE OF ." Not Implemented " TRUE ENDOF
  SWAP FALSE ENDCASE ;
```

```
SCR# 24
\ Build a Term Reference (arg mode)                      LLO 7/20/86

: REF>ARG ( ^term -- )
\ builds an argument from a term reference
  DUP >TYPE VAR.TYPE =          \ check for variable
  IF VAR.TYPE                   \ make a new variable
  ELSE 2@ THEN  PUSH.ARG ;      \ otherwise copy
: VAR>ARG  ( n -- )
\ builds an argument from a variable reference
  COPY.FLG @
  IF NF ELSE CF THEN  +        \ get address of variable
  >ULT.BINDING  REF>ARG  ;     \ get binding for argument
```

```
SCR# 25
\ Build a Structure Record (copy, arg modes)         LLO 7/20/86

: COPY.FUNCT ( arity ^atom -- )
\ build a functor record on the structure stack
  -1 ARG.FLG +!                \ increment the counter
  OVER BYTES/CELL * 4 +        \ compute # of bytes in record
  SS DUP ROT + IS SS           \ allocate space
  DUP 4 +  PUSH.^ARG           \ reset arg pointer for rest
  2! ;                         \ fill head of functor record
```

```
SCR# 26
\ Match a Structure (match mode)                         LLO 7/20/86
\ see FUNCTOR, screen 35
: MATCH.FUNCT ( arity ^atom ^term -- flag )
\ match functor with an argument
\ ARG is reset if functor (name )  and arity match
\ remainder of match is handled by code in clause head
  >POINT                         \ get pointer to record
  DUP STRUC>FUNCTOR ROT =        \ check functor match
   IF DUP STRUC>ARITY ROT =      \ check arity match
    IF STRUC>ARGS PUSH.^ARG TRUE \ reset arg pointer, return T
    ELSE DROP FALSE THEN
   ELSE 2DROP FALSE THEN ;       \ return false if no match
```

```
SCR# 31
\ Prolog Machine Instruction ENTER                        LLO 6/16/86
: RETURN ( -- )
\ Return from a procedure
  ARG.FLG @                         \ check mode
  IF Stack>RP  CF BF U>
   IF CF>NF THEN                    \ reclaim if not backtrack pnt
   Stack>CF                         \ adjust frame pointers
  ELSE StackNF>RP  BF NF =          \ if ret from unit clause
   IF CF>Stack                      \ save parent frame pointer
      RESET.NF                      \ reset frame pointer
   THEN
  THEN ARG.FLG ON                   \ turn off matcher
  INIT.^ARG ;                       \ set arg pointer for next call
```

```
SCR# 32
\ Prolog Machine Instruction  CONST                       LLO 6/16/86
: CONST ( ^atom -- )
\ match or copy a constant
  ARG.FLG @
  IF CONST.TYPE PUSH.ARG                 \ push arg in arg mode
  ELSE ARG&TYPE VAR.TYPE =               \ get first argument
      IF CONST.TYPE SWAP BIND            \ if variable, bind it
      ELSE >POINT = NOT                  \ otherwise must be EQ
       IF IP> DROP    NF BF =
          IF RETRY                       \ retry if new call
          ELSE BACKTRACK THEN            \ backtrack if not
       THEN
      THEN                               \ else continue
  THEN        ;
```

```
SCR# 33
\ Prolog Machine Instruction  VAR
: VAR ( n -- )
  ARG.FLG @
  IF   VAR>ARG                      \ get binding for argument
  ELSE NF + >ULT.BINDING            \ get the variable bindings
   DUP >TYPE  CASE                  \ case analysis on type
     VAR.TYPE   OF UNIFY.VAR   ENDOF
   CONST.TYPE   OF UNIFY.CONST ENDOF
   FUNCT.TYPE   OF UNIFY.FUNCT ENDOF
   ENDCASE      NOT
   IF R> DROP  NF BF =              \ if match not successful
      IF RETRY                      \ retry
      ELSE BACKTRACK THEN           \ or backtrack
   THEN
  THEN ;                            \ build an argument
```

```
SCR# 41
\ Initialization words for test routines                  LLO 6/16/86
: INIT.^ARG.STACK
\ init argument pointer stack
  ^ARG,STACK DUP 2+ SWAP ! ;
: INIT.STACKS
\ initialize stacks prior to CALL
  STRUCTURE.STACK IS SS
  TRAIL IS TS
  CONTROL.STACK IS CF
  CF IS NF  0 IS BF
  ARG.FLG OFF COPY.FLG OFF
  CONTROL.STACK 3000 0 FILL     INIT.^ARG.STACK ;
: TEST INIT.STACKS CR ;
```

```
SCR# 42
\ Test Procedures
\ Create Dictionary Entries for all Test Words

CREATE REV   0 ,        CREATE APPEND 0 ,
CREATE NIL   0 ,        CREATE cons   0 ,
CREATE bob   0 ,        CREATE art    0 ,     CREATE fred 0 ,
```

```
SCR# 43
\ APPEND and REV    (Note arguments of VAR)
1 3 APPEND :ASSERTZ NIL CONST 24 VAR 24 VAR RETURN ;ASSERT
4 3 APPEND :ASSERTZ 2 cons FUNCTOR 24 VAR 28 VAR POP
                    32 VAR
                    2 cons FUNCTOR 24 VAR 36 VAR POP
            ENTER   28 VAR 32 VAR 36 VAR 3 APPEND CALL
            RETURN  ;ASSERT
0 2 REV    :ASSERTZ NIL CONST NIL CONST RETURN ;ASSERT
4 2 REV    :ASSERTZ 2 cons FUNCTOR 20 VAR 24 VAR POP 28 VAR
            ENTER   24 VAR 32 VAR 2 REV CALL
                    32 VAR
                    2 cons FUNCTOR 20 VAR NIL CONST POP 28 VAR
                    3 APPEND CALL
            RETURN  ;ASSERT
```

```
SCR# 44
\ Test of Append
\ use: TEST 0 TEST1 CALL
\ check result with CONTROL.STACK 12 + PRINT.TERM
CREATE TEST1  0 ,
\ append([bob,fred],[art],L).
1 0 TEST1 :ASSERTZ  ENTER
            2 cons FUNCTOR bob CONST
            2 cons FUNCTOR fred CONST NIL CONST POP POP
            2 cons FUNCTOR art  CONST NIL CONST POP
            12 VAR 3 APPEND CALL   RETURN ;ASSERT
```

```
SCR# 38

: PRINT.FUNCT ( ^term -- )
\ print a structure
  >POINT DUP @ BODY> .ID  ASCII ( EMIT
  2+ DUP 2+ SWAP @ BYTES/CELL * OVER + SWAP
  DO I PRINT.TERM BYTES/CELL +LOOP  ASCII ) EMIT ;
: <PRINT.TERM> ( ^term -- )
\ print a Prolog term
  >ULT.BINDING DUP >TYPE CASE
  CONST.TYPE OF PRINT.CONST  ENDOF
    VAR.TYPE OF PRINT.VAR    ENDOF
  FUNCT.TYPE OF PRINT.FUNCT  ENDOF  ENDCASE ;
' <PRINT.TERM> IS PRINT.TERM
```

```
SCR# 39
\ Auxiliary Words                                         LLO 6/16/86
: !CODE.DATA ( nvars arity -- )
\ pack the number of variables (nvars) and arity
\ then enclose
  256 * + ,  ;              \ arity is in high order byte
: \LINK ( start.addr -- end.addr )
\ traverse links to end
  BEGIN DUP @ IF @ FALSE ELSE TRUE THEN UNTIL ;
```

```
SCR# 40
\ PVM to Forth Compiler                                   LLO 6/16/86
: :ASSERTZ ( nvars arity pred -- )
\ add a clause to the Prolog data base
  2DUP FIND.PROC                    \ find proc record
  IF \LINK                          \ if found, get last clause rec
     HERE SWAP ! 0 ,                \ install links
     DROP !CODE.DATA                \ install nvars, arity
  ELSE \LINK                        \ if not found get last proc
     HERE SWAP ! 0 ,                \ install links
     DUP , HERE 2+ ,                \ install arity and link to clause
     0 , !CODE.DATA                 \ set clause link
  THEN STATE ON  ;                  \ compile remainder of input
: ;ASSERT ( -- )
  STATE OFF ; IMMEDIATE             \ turn off compilation
```

```
SCR# 34
\ Prolog Machine Instruction  POP                         LLO 7/20/86

: POP ( -- )
\ pop from a FUNCTOR
  POP.^ARG                          \ restore argument pointer
  ARG.FLG @                         \ look for "arg" mode
  IF 1 ARG.FLG +! THEN ;            \ decrement counter
```

```
SCR# 35
\ Prolog Machine Instruction  FUNCTOR                     LLO 7/20/86
: FUNCTOR ( arity ^atom -- )
\ Compiler object indicating a structure
  ARG.FLG @
  IF SS FUNCT.TYPE PUSH.ARG   COPY.FUNCT
  ELSE ARG&TYPE CASE
     VAR.TYPE OF MATCH.VAR   ENDOF
   FUNCT.TYPE OF MATCH.FUNCT ENDOF
   FALSE SWAP ENDCASE
   NOT IF R> DROP NF BF =
        IF RETRY ELSE BACKTRACK THEN
       THEN
  THEN ;
```

```
SCR# 36
\ Support Routines
\
\ ASSERTZ is the PVM to Forth Compiler.
\ It requires as parameters the number of variables
\ in the clause, the arity of the clause, and the pfa
\ of the clause functor.
\ If the PVM word set were extended, with different words
\ for instructions in the head and body, much of the compilation
\ to the extended word set could be handled by ASSERTZ, since
\ it can tell the difference between the head and body of a
\ clause.  Thus the Prolog-PVM compiler could stay simple.
\
```

```
SCR# 37
\ Prolog Print Words
  DEFER PRINT.TERM
: PRINT.CONST ( ^const -- )
\ print a constant
  >POINT BODY> .ID ;
: PRINT.VAR ( ^var -- )
\ print a variable
  >POINT BASE @ >R
  ASCII _ EMIT HEX U.    R> BASE ! ;
```

Suite et Fin au page 16

PROJET F32
SPECIFICATION PRELIMINAIRE

par Paul ORTAIS

Contact: 3615 SAM*JEDI Bal FF32.

I. POINT DE DEPART

Le projet F32 est issu d'une triple rencontre:

- l'émergence d'une classe nouvelle de processeurs universels, dits "machines de piles", particulièrement simples et puissants. L'état actuel des machines de piles mérite des améliorations.

- le développement du langage FORTH, dédié à la manipulation desdites piles, à l'exécution duquel ces machines sont dédiées. Le potentiel de ce langage mérite néanmoins d'être mis en valeur.

- les moyens de réalisation de chips VLSI à notre disposition permettent d'entreprendre à peu de risque une implémentation performante qu'il nous reste maintenant à décrire.

On verra à la lecture de ce premier document que la voie de recherche n'est pas univoque, mais ramifiée; on découvre plutôt un ensemble de directions possibles, parmi lesquelles il faut trancher.

C'est pourquoi, d'une part F32 nécessite une constitution d'équipe, d'autre part cette spécification doit être comprise comme une proposition initiale, un moyen d'initialisation.

Trois axes de réflexion sont développés ici:

- le VLSI par lui-même, dont la réalisation est l'aspect le plus simple du projet,

- l'orientation du processeur vers la manipulation des bases de données, et pourquoi ceci constitue non un luxe d'implémentation, mais un passage obligé vers un débouché du produit,

- ce que devrait être une implémentation minimale, permettant ensuite de finaliser le processeur en observant le comportement de ce noyau.

I.1 Processeur.

A l'origine, le NC4016 fût développé en 85 par des fans du Forth voulant créer une machine-langage spécifique. Le résultat, malgré un succès commercial limité, fût un processeur universel de très grande puissance et très simple. Son apport tient à l'utilisation de mémoires physiques distinctes pour implémenter les espaces logiques de pile de donnée et pile de retour.

Ces espaces existent dans tout langage, mais ils y sont presque toujours implicites et sous-entendus. Dans le FORTH, leur manipulation explicite a conduit naturellement à leur prise en considération dès le design du NC4016.

Il apparût alors que, tout le changement de contexte devenant un opérateur de base, le processeur est déchargé du principal goulot d'étranglement des µP modernes. C'est ce problème qui est résolu, avec beaucoup moins d'élégance, par toute la génération des RISCs actuels.

De même, la pile de paramètres fournit un second bus de données, le gain résultant est analogue à ce qui se fait en modèle de Harvard.

Par extension, on voit que la puissance est (beaucoup) une question de nombre de pins disponibles. Cependant, l'accroissement est exploitable aisément par une machine de pile, et très peu autrement...

Un nouvel avatar du NC4016, le RTX2000 d'Harris, reprend le même noyau en l'habillant de périphériques qui en font un composant puissant, qui fera probablement une brillante carrière dans le "temps réel". Néanmoins, il n'apporte rien de neuf conceptuellement dans le cadre de la présente étude.

I.2 Langage:

Qui dit processeur dit langage. Bien que rien ne doive en principe forcer tel ou tel langage sur un processeur, il y a toujours de tels appariements, ne serait-ce qu'historiquement. La machine de pile se prête bien au FORTH, mais on verra que l'implémentation rigoureuse du langage conduit à des pertes d'efficacité.

Le C comme tous les langages procéduraux tombe dans une classe qui justifie l'écriture de compilateurs classiques, laquelle est hors sujet à ce stade du projet.

I.2.1 LISP

Il faut étudier les mécanismes de LISP pour saisir l'approche base de données du F32. LISP est pertinent pour l'approche ouverte des structures qu'il fournit aux manipulations de données.

Cependant, LISP est lent. Dans F32, à chaque fois qu'il faut choisir entre la puissance et un autre aspect, c'est la puissance qui doit l'emporter. Et la puissance, c'est le FORTH.

I.2.2 FORTH

Conçu à l'origine par Charles MOORE comme un macro-langage de gestion de fichiers à usage personnel, comparable au DCL de DEC, le FORTH est caractérisé par sa constitution incrémentale.

Pour y parvenir de façon simple (la simplicité dans cet univers est une vertu majeure), il utilise une notation postfixée qui permet une syntaxe incrémentale exempte de Parser de commande.

Secondement, le langage est bâti sur un "dictionnaire", liste de procédures (mots) ou chaque nouvelle entrée appelle les procédures précédemment définies.

Il n'y a qu'un seul type reconnu, le mot. Il vaudrait mieux parler d'objet, bien que cette notion soit postérieure au FORTH. Le mot désigne autant les variables, constantes, pointeurs, procédures et structures.

Le moteur interne du langage est le plus simple qui soit:
- incrémenter le pointeur de programme d'un mot courant,
- si on trouve une valeur, l'empiler,
- si on trouve un mot du vocabulaire, l'exécuter,
- boucler.

Le point intéressant du langage est l'extrême simplicité du noyau, ce qui permet de commencer l'utilisation du processeur avec un investissement logiciel vraiment minimal. C'est ce qu'il nous faut.

Dans le cadre de notre propos, disons que FORTH est un langage en kit, livré très simple, avec lequel on peut tout faire, ou bien on doit tout faire, suivant que l'on est optimiste ou pessimiste.

D'un abord plutôt rébarbatif au départ, sa grande simplicité permet de l'apprécier assez vite. Bien que F32 ne soit pas un projet dédié au FORTH, la compréhension du langage est vivement recommandée.

Pour consoler le lecteur, la littérature est disponible et claire; l'environnement est accessible et gratuit; enfin la communauté des utilisateurs de FORTH est active, ouverte et compétente en France.

Quoiqu'il en soit, la connaissance pratique du langage est impérative si l'on veut optimiser le design.

Pour finir, il est tout à fait possible que la recherche de structuration du FORTH, menée avec F32, n'aboutisse pas, ou bien soit nettement plus longue que le développement VLSI.

Dans ce cas, il faut se concentrer sur une recherche de puissance pure, exploitant les avantages connus de la combinaison FORTH-machine de pile.

I.3 Environnements logiciels.

A moins de remaniements très sérieux, le FORTH ne sera pas le langage de choix pour de tels environnements. Dans sa forme actuelle, il a une forme trop primitive pour autoriser les très grosses applications courantes dans ce domaine. Néanmoins, il faut remarquer que:

- la compacité propre au FORTH est une résistance à l'inflation du code, phénomene par ailleurs courant et qui indique plutôt une mauvaise comprehension des traitements et contrarie une exécution optimale. En particulier, le système devient essentiellement dépendant des accès disque pour ce qu'un Forth exécuterait surtout en RAM. La meilleure compacité du Fort croit avec la taille de l'application. Le noyau du langage est si compact qu'il tient entièrement à l'intérieur d'un jeu d'instruction.

- FORTH est la seule approche permettant une vraie compilation en temps réel.

- structurer le noyau Forth en noyau de base de données hiérarchisée semble relativement simple. Voir aussi l'implémentation nommée Le-Forth.

- même si Forth n'est pas le langage cible finalement préféré, il reste le meilleur moyen de valider le design, pour sa simplicité.

I.4 Usage des processeurs.

Ces dernière années, les capacités d'intégration disponibles autorisent la réalisation en semi-custom de CPU 32 bits. Parallèlement, ce format tend à se vulgariser. Il est donc naturel de se demander si une machine de pile en 32 bits est envisageable. Les seules instances actuelles sont en 16 bits (NC4016/RTX2000). Leur niveau de performance est comparable aux meilleurs CPU 32 bits actuels. Que peut-on attendre en 32 bits?

Il apparait dès à présent que ce chip sera nettement plus aisé à mettre en oeuvre que ses analogues. On doit donc privilégier les champs d'applications de mise en oeuvre coûteuse en temps, comme le contrôle temps réel et le génie logiciel:

- dans le premier cas on bénéficie de la transparence de la machine virtuelle pour faciliter la mise au point matérielle des systèmes.

- dans le second, la machine de pile apporte une exécution plus rapide des chainages, et la possibilité d'un traçage matériel du procédé par observation des piles, ce qui n'est pas envisageable sur une machine classique.

Tout d'abord, ce format implique un large espace adressable. Les applications exploiteront des bases de données, que ce soit en IA, CAO, génie logiciel, gestion de base, imagerie, etc...

On doit se demander si le gain en puissance observé en 16 bits se retrouvera en 32.

C'est un objectif majeur de F32 de pousser l'avantage du chip vers la manipulation de bases de données.

II. DEVELOPPEMENT.

Une grande part est encore spéculative, néanmoins on peut fixer les grandes lignes suivantes:

la structure du NC4016 doit être bien maitrisée, il s'agit là de la référence en la matière. F32 devrait différer nettement de ce modèle, qui doit cependant être toujours en vue.

Initialement, on bâtira une structure minimale, c'est à dire qu'on évitera les optimisations coûteuses en temps de développement.

L'objectif est d'abord de créer une machine de base robuste et de structure régulière. On vérifiera le gain de ce premier stade par l'exécution de benchmarks classiques, qui constituent un jeu de tests suffisants et mesurent précisément les qualités du design.

La partie noble du projet concerne l'unité de gestion de base de donnée associée au core, celui-ci devant être vu comme apportant la puissance brute au service de cette unité.

II.1 Machine de pile.

ON aura compris que F32 est une machine de pile... Physiquement, l'unite centrale utilise plusieurs voies de données externes vers les espaces suivants:

- mémoire principale, comme tout processeur.

- pile de données, ou pile paramètre, stockant les données locales des procédures en cours.

- pile de retour, gérant les redirections du processeur, depuis les boucles jusqu'aux exceptions.

- pile annexe, si matériellement elle est possible.

L'accès à ces espaces est soit explicite, comme c'est souvent le cas de la mémoire principale, ou masqué comme presque toujours pour la pile de retour. Cependant, dans tous les cas, on conserve une possibilite d'accès explicite.

II.2 Langage.

Comme on l'a vu, le langage qui aura la préférence au niveau de l'émulation sera FORTH. Au niveau simulation de base, se sera sûrement les commandes du simulateur logique des outils, mais il est bon que ce stade soit minimisé.

L'idéal est un interfaçage FORTH générant du source F32 pour l'émulation. Cet interfaçage est déjà quasiment opérationnel, aux opcodes près; il est dû à JEDI, le "FRench Forth Interest Group". Cet interface permet d'émuler F32 avant design sur quasiment tous les micro-ordinateurs actuels.

Pour partitionner, disons que l'émulateur FORTH permettra de simuler F32 en partant du niveau où le simulateur commence à saturer, à partir de quelques centaines d'instructions.

II.3 Opérateurs de base de données.

L'idée de base est que la présentation naturelle de l'univers est hiérarchisée et non linéaire (=Cartésienne), ce dont tout le monde convient aisément; que ladite présentation est la même pour un process quelconque, en particulier pour toute application exécutée par un processeur quel qu'il soit.

Il se trouve que dans des cas simples, on peut assimiler les objets manipulés à des structures simples telles que des tableaux, des variables et du texte. Le cas n'est particulier qu'en apparence: en réalité, l'arbre est ici réduit à ses composants terminaux, qui seront toujours de tels objets élémentaires. Même dans ce cas, une bonne part de l'application consiste à créer une structure pour manipuler les objets en question.

Or, si l'on avait disposé au départ d'un outil de description hiérarchisé, ce qui est, autrement, long à décrire et masqué, il aurait été immédiat et explicite. Par exemple, sous cette pproche, il y a peu de différence entre les descripteurs d'une application à menus déroulants, un tableur, le simulateur d'un design électronique ou mécanique, etc...

L'obstacle principal au développement du logiciel réside dans le masquage des structures implicites, qui rend la description des applications perceptible à leur seul auteur, et souvent pour un temps limité. Le développement du matériel s'est bâti sur des structures objectives, c'est à dire admises de tous; un design est accessible à tous, et le reinvestissement des efforts possibles.

Pour développer des applications d'appréhension délicate, il a fallu créer des langages dits d'intelligence artificielle, destinés à décrire explicitement les structures plus complexes.

En particulier la distinction entre les structures manipulées et celles du langage s'estompent et parfois disparaissent (LISP).

Ceci dit, l'exécution de ces langages reste lourde, c'est à dire lente. On y pallie en construisant des processeurs spécifiques, mais aucun jusqu'à présent n'a pu quitter sa niche.

La raison semble être que la restriction à un langage donné ôte à un processeur la possibilité de durer sur le marché.

En conséquence, la fonction de gestion de base de donnée doit etre câblée, mais sur une machine d'usage général.

Dans la pratique que doit faire le chip? Il doit avoir le sens de l'orientation à l'intérieur d'un graphe. Par exemple, si l'on décrit l'espace de l'application comme un arbre, au moyen d'une liste de type LISP, F32 possèdera deux modes d'adressage spéciaux:

- absolu, avec au départ un pointeur de racine d'arbre et un pointeur de path (chemin) et le processeur trouve dans la liste l'objet pointé par Path; il empile aussi les paramètres utiles du point courant, à savoir les pointeurs Père, sous-cellules, type de liste courante, type d'objet atteint, etc...

- relatif, le point de départ n'étant pas forcément la racine. L'intérêt est ici la vitesse, F32 pouvant évaluer en pipe-line une liste au débit (DMA) d'un mot de 32 bits par 80 ou 60 ns. On épluche un arbre de quelques milliers de cellules dans le temps que met un 80286 à changer de contexte.

On est bien obligé de privilégier une structure à priori pour les listes de base, et donc faire le pari de leur fécondité comme support d'usage général.

L'option consiste à faire des listes de pointeurs de 32 bits, alignés sur un octet faible multiple de 4, pour acquérir la liste avec le plus fort débit. Le digit de poids fort est un Tag décrivant la fonction dans la liste du pointeur.

Les Tag de base sont au nombre de trois seulement:

- descente d'un niveau dans la hiérarchie "(" en édition
- montée d'un niveau ")"
- incrément de rang de liste ","

L'éditeur d'arbre fait apparaître ...nnn1, nnn2, nnn3(aaa,bb))... ce qui équivaut à dire qu'il y a dans l'espace de l'application une liste contenant successivement les objets en [nnn1] puis en [nnn2] puis [nnn3]. On voit que l'objet en [nnn3] est fait de ceux pointés par aaa et bb. Pour savoir ce qu'il y a en nnn3 et pourquoi on y a besoin de ceux qui est pointé par aaa et bb, je pense que le mieux est d'aller voir sur place. Par bonheur, F32, arrivé là, se retrouve pointé sur nnn3.

Si le mécanisme est assez pratique et rapide, le programmeur sera tenté de décrire le maximum de son application en n'utilisant que l'éditeur d'arbre (le BROWSER) et ces modes d'adressage symbolique.

La plus grande ambition de F32 est là, et aussi de créer un noyau FORTH compatible avec ce mécanisme de recherche.

II.4 Système d'exploitation.

Il est vrai qu'un processeur est d'usage général s'il peut s'accommoder de tout environnement, moyennant l'écriture des compilateurs raisonnablement complexes.

Il faut dire néanmoins que le produit est mort-né s'il ne "tourne" pas sous un systeme d'exploitation répandu à sa sortie. Harris en fait actuellement l'amère expérience avec le RTX2000 qui a été diffusé six mois avant son langage C. L'interface très primitif des premiers échantillons, un vieux Forth assez obscur, a déjà rebuté plus d'un client potentiel.

Un premier refus du marché est extrêmement difficile à remonter. Dans le cas de RTX2000, il eût fallu délivrer un interface très collé à MS-DOS, ne divergeant qu'au niveau de l''assembleur à moins que l'utilisateur ne désire utiliser FORTH directement.

II.4.1 UNIX.

Il s'agit de ne pas faire la même erreur avec F32. Même si le core doit être incrusté dans des ASIC, ce sera bien une machine de pile à programmer. Le système d'exploitation évident est UNIX. Qui peut nous porter un FORTH 32 bits sur UNIX?

Un tel développement devrait démarrer en parallèle avec l'implémentation hard. Les station Apollo sont particulièrement indiquées pour cela.

■

Suite de la page 13

```
SCR# 45
\ Test of REV
\ use: TEST 0 TEST2 CALL
\ check result with CONTROL.STACK 12 + PRINT.TERM
CREATE TEST2 0 ,
\ rev([bob,art,fred],L).
1 0 TEST2 :ASSERTZ  ENTER
            2 cons FUNCTOR bob CONST
            2 cons FUNCTOR art  CONST
            2 cons FUNCTOR fred CONST NIL CONST
                   POP POP POP
            12 VAR 2 REV CALL RETURN ;ASSERT
```

■

DRAGON

von Christoph Krinninger

Dieses Programm zeichnet den sogenannten Harter-Hightway-Drachen. Es benutzt die hohe Auflösung des Atari ST und die hohe Geschwindigkeit des volksFORTH und bietet während des Aufbaus eine ganz besondere Ästhetik.

Für den Aufbau wird eine sogenannte Turtle verwendet, dieser muß nur eine Drehrichtung und eine Zeichenlänge mitgeteilt werden und ist dadurch eine besonders unkompliziertes Zeichenwerkzeug. Für die Umsetzung dieser Vektorgrafik in die Bildschirmkoordinaten werden elementare trigonometrische Funktionen wie Sinus und Cosinus benötigt. Wenn keine besonders hohe Auflösung benötigt wird, so liest man diese Werte am einfachsten in einer Tabelle ab, wobei man nicht unbedingt für jeden Winkel zwischen 0 und 360 Grad einen Wert abspeichern muß. Es genügt nach den bekannten Umformungsformeln durchaus der Bereich zwischen 0 und 90 Grad. Ferner kann man weiteren Speicherplatz einsparen, indem man nur die Werte für gerade Winkel abspeichert und für ungerade Winkel linear extrapoliert.

Die DRAGON-Kurve besteht aus rechtwinkligen Linien, die mit Hilfe der Rekursion in immer kleinere Abschnitte geteilt werden. Das Programm wird mit N DCURVE gestartet, N ist die Ordnungszahl. Mit der Auflösung des Atari ST kann man maximal einen Drachen 16. Ordnung zeichnen. Die Rechenzeit beträgt ca. 4 Minuten. Mit der Variablen STEPSIZE kann man die Größe der Grafik verändern.

Diese Kolumne soll zu einer ständigen Einrichtung werden. Wer also eine FORTH-Grafik erstellt hat, die für das Titelbild geeignet ist, möge einen kurzen Artikel schreiben und diesen an die Redaktion senden.

Stichworte:

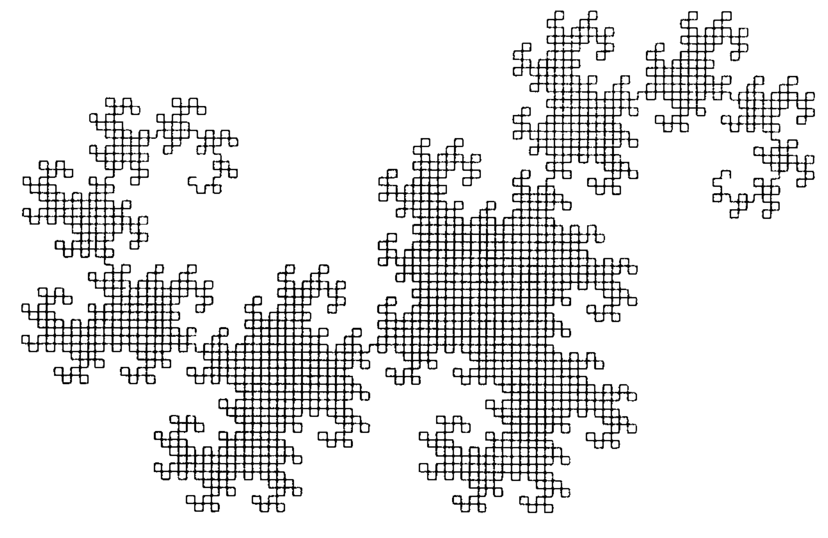
- » Fraktale,
- » Turtle-Grafik,
- » Trigonometrie

Noch ein paar Worte zum Artikel in der letzten Ausgabe:

Durch einige Besonderheiten des Ventura Publisher sind bei diesem Artikel alle ">" und "<" unterdrückt worden. Dies ist besonders bei FORTH-Worten wie ">R", "R>" oder "->" verheerend. Wer am korrekten Sourcecode oder Artikel interessiert ist, kann sich mit mir über das FORTH-Büro in Verbindung setzen.

Bibliographie:

1. DRAGON, Bruce R. Land, BYTE 04/86, S. 137 ff
2. SIN, COS und 3D, Frank Schmidt, c't 10/85, S. 87ff
3. The Fractal Geometry of Nature, Mandelbrot B. B., Freeman 1982, S. 66ff



Beispiel 1

Screen # 1

```
\ Loadscreen                                  21feb88 ck

Onlyforth   gem also

\needs it       : it ;
forget it       : it ;

\needs pline   include vdi.scr

2 8 thru
```

Screen # 2

```
\ Sinustabelle                                21feb88 ck

decimal

Create sintab

0 ,  349 ,  698 ,  1045 ,  1392 ,  1736 ,  2079 ,  2419 ,
2765 ,  3090 ,  3420 ,  3746 ,  4067 ,  4384 ,  4695 ,  5000 ,
5299 ,  5592 ,  5878 ,  6157 ,  6428 ,  6691 ,  6947 ,  7193 ,
7431 ,  7660 ,  7880 ,  8090 ,  8290 ,  8480 ,  8660 ,  8829 ,
8988 ,  9135 ,  9272 ,  9397 ,  9511 ,  9613 ,  9703 ,  9781 ,
9848 ,  9903 ,  9945 ,  9976 ,  9994 , 10000 ,
```

Screen # 3

```
\ Sinusberechnung                             21feb88 ck

: ((sin ( n1 -- n2 )
dup dup 2/ 2* =
IF
sintab + @
ELSE
sintab + 1- dup @ swap 2+ @ + 2/
THEN ;

: (sin  ( n1 -- n2 )
dup 180 > IF  180 - true  ELSE  false  THEN
swap dup 90 > IF  negate 180 +  THEN
((sin
swap IF  negate  THEN ;
```

Screen # 10

21feb88 ck

Kleiner Trick, um das Programm wiederholt hintereinander laden zu können.

Screen # 11

21feb88 ck

SINTAB   Tabelle für Sinus-Berechnung

Screen # 12

21feb88 ck

((SIN   Sinus-Berechnung für Winkel zwischen 0 und 90 Grad.

Gerade Winkel

Ungerade Winkel
Näherung nach sin(x) = ((sin(x-1) + sin(x+1))/2

(SIN   Sinus-Berechnung für Winkel zwischen 0 und 360 Grad.

Screen # 4

```
\ Sinus, Cosinus                         21feb88 ck

: sin   ( n1 -- n2 )
dup 0<   IF BEGIN 360 + dup   0> UNTIL THEN
dup 360 > IF BEGIN 360 - dup 360 < UNTIL THEN
(sin ;

: cos  ( n1 -- n2 )
90 + sin ;
```

Screen # 13

21feb88 ck

SIN Sinus-Berechnung für beliebige Winkel

COS Cosinus-Berechnung nach der Formel
cos(x) = sin(90 + x)

Screen # 5

```
\ Turtle-Grafik                          21feb88 ck

Variable angle
Variable xcood
Variable ycood
Variable stepsize

Variable #lines      25 Constant #maxlines

Create line-array    #maxlines 4 * allot

: turn  ( deltaangle -- )
angle +! ;
```

Screen # 14

21feb88 ck

ANGLE Einige Variable für die Turtle-Grafik
XCOOD
YCOOD
STEPSIZE Länge eines Turtle-Schrittes

#LINES #MAXLINES Zähler für die Akkumulation der Poly-linie.
LINE-ARRAY Zwischenspeicher für Linienkoordinaten

TURN Dreht die Turtle gegen den Uhrzeigersinn

Screen # 6

```
\ Turtle-Grafik mit Akkumulation          21feb88 ck

: draw-line    ( -- )
#lines @ 0
DO line-array I 4 * + 2@ LOOP
#lines @ [ gem ] pline
xcood @ ycood @ line-array 2! 1 #lines ! ;

: move-turtle   ( -- )
stepsize @ dup
angle @ cos 10000 */ xcood @ + dup xcood ! swap
angle @ sin 10000 */ ycood @ + dup ycood !
line-array #lines @ 4 * + 2!
1 #lines +!
#lines @ #maxlines < not IF draw-line THEN ;
```

Screen # 15

21feb88 ck

DRAW-LINE Zeichnet die zwischengespeicherte Poly-Linie

MOVE-TURTLE Bewegt die Turtle um den Betrag in STEPSIZE und speichert die Bewegung.

Screen # 7

```
\ Hauptroutine                              21feb88 ck

: dragon        ( sign level -- ) recursive
key? IF key abort THEN
dup 0=
IF
drop drop move-turtle
ELSE
over 45 * turn
1
over 1- dragon
over -90 * turn
-1 ( +1 ergibt eine weitere hübsche Grafik )
over 1- dragon drop 45 * turn
THEN ;
```

Screen # 8

```
\ Initialisierung                           21feb88 ck

: dcurve        ( level -- )
page  200 xcood ! 150 ycood !  360 6 * angle !
2 stepsize !
xcood @ ycood @ 2dup 2 pline  1 swap
overwrite solid
xcood @ ycood @ line-array 2!
1 #lines !
dragon
#lines @ 0<> IF draw-line THEN ;
```

Screen # 16

```
21feb88 ck

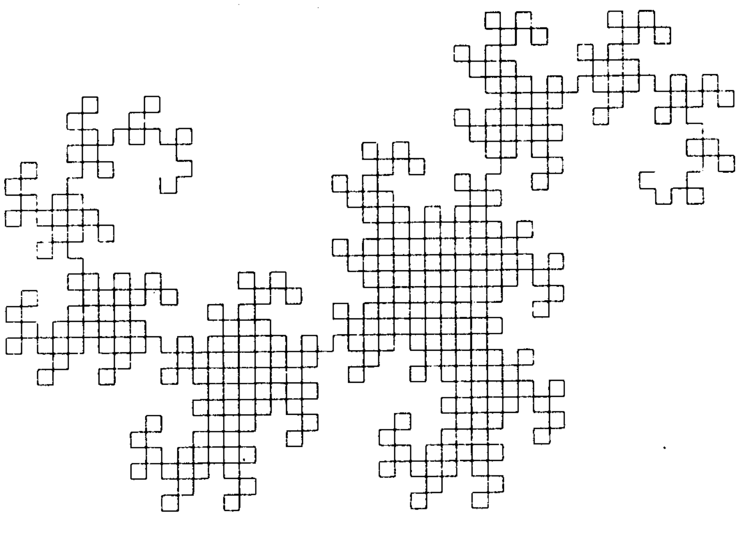
DRAGON  Eigentliche Hauptroutine
```

Screen # 17

```
                                            21feb88 ck

DCURVE  Zeichnet die DRAGON-Kurve n-ter Ordnung.
N muß geradzahlig sein.

Maximal ist ein Drachen 16. Ordnung bei STEPSIZE=1
auf dem Atari ST möglich.
```



Beispiel 2